GUILHERME KLEIN GOMES

CONTROLE PREDITIVO EM TEMPO-REAL PARA SEGUIMENTO DE TRAJETÓRIA DE VEÍCULOS AUTÔNOMOS

FLORIANÓPOLIS
2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

CONTROLE PREDITIVO EM TEMPO-REAL PARA SEGUIMENTO DE TRAJETÓRIA DE VEÍCULOS AUTÔNOMOS

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

GUILHERME KLEIN GOMES

Florianópolis, Março de 2006.

CONTROLE PREDITIVO EM TEMPO-REAL PARA SEGUIMENTO DE TRAJETÓRIA DE VEÍCULOS AUTÔNOMOS

Guilherme Klein Gomes

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em *Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.

Prof. Leandro Buss Becker, Dr.
Orientador

Prof. Alexandre Trofino Neto, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Santa Catarina

Banca Examinadora:

Prof. Leandro Buss Becker, Dr.
Orientador

Prof. Julio Elias Normey-Rico, Dr.
Co-orientador

Prof. Edson De Pieri, Dr.

Prof. Raul Guenther, Dr.

Prof. Jean-Marie Farines, Dr.

Prof. Christian Roberto Kelber, Dr.

Ao meu filho Lorenzo e a minha esposa Bianca.

AGRADECIMENTOS

Agradeço ao meu filho Lorenzo e à minha esposa Bianca, meus maiores motivos de felicidade, pelo apoio nos momentos difíceis e pela compreensão durante os longos períodos de ausência.

Agradeço a meus pais Luiz Fernando e Heloísa, pela ajuda nos momentos mais difíceis, e aos meus sogros Rosaura e Gian, pela disposição e apoio em tempo integral, procurando sempre suprir a minha ausência com relação ao Lorenzo e à Bianca.

Agradeço aos meus grandes amigos Zeca, Nisus, Doti, Raquel, Lízia e família, todos moradores de Florianópolis, pela hospitalidade e amizade. Zeca, Nisus e Doti, as festas e as sessões de surfe no Santinho e em Garopaba ficarão para sempre na memória.

Agradeço em especial ao meu orientador, Prof. Leandro Buss Becker, pelo apoio e dedicação no desenvolvimento desta dissertação, além da amizade, parceria e companheirismo nos churrascos e sessões de surfe.

Ao meu co-orientador, Prof. Julio Elias Normey Rico, pelos ensinamentos e disponibilidade, não apenas durante o desenvolvimento da dissertação, mas também durante a disciplina de Fundamentos de Controle, além da amizade.

Ao Prof. Christian Roberto Kelber, pelos ensinamentos e apoio junto à UNISINOS, e principalmente por ter me despertado a motivação pela área de veículos autônomos.

Agradeço em especial também ao Guilherme, pela amizade, parceria e pela grande colaboração no desenvolvimento deste trabalho.

Agradeço aos membros da banca examinadora pela contribuições e sugestões, e ao Programa de Pós-Graduação em Engenharia Elétrica, pela oportunidade de realização do mestrado e desenvolvimento da pesquisa junto à UFSC.

Por fim, agradeço a todos os amigos que fiz durante o curso de mestrado, em especial ao Marcelo Coutinho e aos colegas do Laboratório de Controle e Micro-Informática, pelo companheirismo e auxílio ao longo de todo o mestrado. Coutinho, obrigado pelas músicas e pelas cópias da dissertação.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

CONTROLE PREDITIVO EM TEMPO-REAL PARA SEGUIMENTO DE TRAJETÓRIA DE VEÍCULOS AUTÔNOMOS

Guilherme Klein Gomes

Março/2006

Orientador:	Prof. Leandro Buss Becker, Dr.
Área de Concentração:	Automação e Sistemas.
Palavras-chave:	veículos autônomos, controle preditivo, seguimento de trajetórias, sistemas tempo real.
Número de Páginas:	178

Neste trabalho é realizado o estudo e a implementação de um sistema tempo-real embarcado para realizar o controle de seguimento de trajetória em veículos autônomos. O sistema desenvolvido emprega técnica de controle preditivo baseado em modelo, sendo destinado ao controle do comportamento cinemático e também do comportamento dinâmico do veículo. O controle do comportamento cinemático é baseado em modelo linearizado utilizando sistema de coordenadas locais e técnica de geração de trajetórias de aproximação, buscando uma condução suave e factível. O controle do comportamento dinâmico é destinado a aplicações de alto desempenho. O projeto e a implementação do sistema em questão foram desenvolvidos com base no paradigma de orientação a objetos visando a obtenção de código modular e de fácil manutenção, e que apresente estrutura lógica favorável à reutilização de software e à portabilidade. Realizou-se também uma análise criteriosa das restrições temporais do sistema, por ser este um sistema de tempo-real. Através de ensaios experimentais em diferentes plataformas comprova-se a eficiência do sistema embarcado desenvolvido, tanto em termos das técnicas de controle como da estruturação da aplicação.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

REAL-TIME PREDICTIVE CONTROL FOR PATH FOLLOWING OF AUTONOMOUS VEHICLES

Guilherme Klein Gomes

March/2006

Advisor: Prof. Leandro Buss Becker, Dr.

Area of Concentration: Automation and Systems.

Keywords: autonomous vehicles, predictive control, path following, real-time systems.

Number of Pages: 178

This work presents design and implementation of a real-time embedded system for path following on autonomous vehicle applications. The system uses Model Based Predictive Control algorithm and can be applied to control the kinematic and the dynamic behavior of autonomous vehicles. The kinematic behavior control is based on a linearized model using local frame coordinates and approaching path, to get smooth and feasible guidance. The dynamic behavior control is destined to high performance applications. The system's design and implementation process is based on the object-orientated paradigm, aiming for modularity, maintainability, extensibility, and reusability of the system software. Furthermore, the design process was driven to design the system in a platform-independent manner. Due to real-time caracteristics, detailed time constrains analysis are also performed. Experimental tests were performed to validate the system's efficiency regarding the control technique as far as the application structuring.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

ABS	*Anti-Blocking System*
AGV	*Autonomous Guided Vehicle*
API	*Application Program Interface*
ARIMA	*Auto-Regressive and Integrated Moving Average*
CARIMA	*Controller Auto-Regressive and Integrated Moving Average*
CPBM	Controle Preditivo Baseado em Modelo
DMC	*Dynamic Matrix Control*
EHAC	*Extended Horizon Adaptive Control*
EPSAC	*Extended Prediction Self Adaptive Control*
ESP	*Electronic Stability Program*
GPC	*Generalized Predictive Control*
GPS	*Global Positioning System*
KDVA	Kit de Desenvolvimento para Veículos Autônomos
MAC	*Model Algorithm Control*
MIMO	*Multiple Input Multiple Output*
OO	Orientado(a) a Objeto
PFC	*Predictive Functional Control*
POO	Projeto Orientado a Objeto
QPF	*Quintic Polynomial Fit*
RMA	Robô Móvel Autônomo
SCST	Sistema de Controle de Seguimento de Trajetória
SISO	*Single Input Single Output*
SP	*Smith Predictor*
TCS	*Traction Control System*

UPC	*Unified Predictive Control*
WCET	*Worst Case Execution Time*
WMR	*Wheeled Mobile Robot*

Capítulo 1

Introdução

1.1 Motivação

Robôs móveis autônomos têm sido utilizados nas mais variadas atividades que vão desde tarefas simples, como aspiração de pó e entrega de correspondência, até tarefas complexas, como exploração planetária e exploração de petróleo e gás natural (OLLERO e HEREDIA, 1995; de OLIVEIRA, 2001). De maneira geral, robôs móveis são utilizados em aplicações destinadas à realização de tarefas em ambientes inóspitos ou onde o homem tem dificuldades ao realizá-las (LAGES, 1998). Em especial, a área de automatização de transporte em rodovias tem motivado muitas pesquisas, nas quais se incluem os sistemas de navegação autônoma para veículos comerciais (AMIDI, 1990; OLLERO e AMIDI, 1991; WIT *et al.*, 2004; LEDUR, 2003; GOMES, 2003).

Entre os diferentes tipos estudados de robôs móveis dotados de rodas, dois modelos específicos têm motivado muitos trabalhos. O primeiro consiste nos robôs dotados de duas rodas não direcionáveis, que realizam curvas através da diferença de velocidade destas rodas, sendo, neste trabalho, denominados *diferenciais*. O segundo modelo consiste em robôs e veículos baseados na Estrutura de Ackerman, também conhecidos como tipo automóvel, com quatro rodas sendo o par dianteiro direcionável, neste trabalho denominados *direcionais*. Estes dois modelos apresentam, além de possíveis limitações físicas (elétricas e mecânicas), restrições que não podem ser integradas para serem descritas exclusivamente em função das variáveis de configuração (MURRAY e SASTRY, 1993), o que os caracterizam como sistemas não-holonômicos. Geometricamente, uma consequência natural destas restrições é a inacessibilidade de alguns pontos do plano de deslocamento, como por exemplo veículos do tipo automóvel, que não são capazes

de girar sobre seu eixo sem se deslocar no plano. Devido a estas restrições, e por serem sistemas não-lineares, robôs e veículos diferenciais e direcionais têm em seu controle um dos principais problemas, tornando a concepção de estratégias de controle para estes sistemas uma tarefa de grande complexidade (KÜHNE *et al.*, 2004).

Estas dificuldades vêm motivando o estudo e o desenvolvimento de diversas técnicas para controle e sensoreamento de dispositivos móveis. Foram desenvolvidos sistemas de navegação executando tarefas de planejamento e geração de trajetórias, localização do veículo dentro do ambiente, aquisição de informações do próprio ambiente e execução do movimento desejado (WIT *et al.*, 2004).

A implementação destes sistemas em veículos autônomos implica a necessidade de unidades de processamento local e dedicadas, constituindo os chamados sistemas eletrônicos embarcados (WOLF, 2001). O desenvolvimento de sistemas embarcados desta natureza tem se demonstrado uma tarefa complexa, devido a necessidades de comportamento flexível e adaptativo, arquitetura distribuída e operação em ambientes hostis, além do respeito a rigorosas restrições temporais. Técnicas convencionais para o desenvolvimento de sistemas computacionais não são capazes de lidar com a crescente complexidade deste tipo de aplicação, principalmente pelo fato de apresentarem pouca capacidade de abstração (BECKER e PEREIRA, 2002). Dentre as técnicas atuais para suprir estas necessidades (AWAD *et al.*, 1996; SELIC *et al.*, 1994), destacam-se o paradigma de orientação a objeto (OO) e, em especial, os chamados *real-time distributed objects* (IEEE, 2000).

Apesar de existirem diversas pesquisas para resolver o problema de seguimento de trajetória de robôs móveis autônomos, a maioria deles trata apenas de algoritmos para o controle do comportamento cinemático e poucas vezes faz referências detalhadas à implementação do sistema. Nestas aplicações, além da necessidade de se considerar o modelo dinâmico (BOYDEN e VELINSKY, 1994), é de fundamental importância a análise criteriosa das restrições temporais do sistema, além da utilização de uma metodologia de desenvolvimento bem estruturada ao longo de todo o ciclo de vida do sistema, compreendendo as etapas de análise, projeto, implementação, testes e manutenção (BECKER e PEREIRA, 2002).

Neste trabalho, propõe-se o desenvolvimento de um sistema tempo-real embarcado

para controle de seguimento de trajetória de rôbos móveis, empregando técnicas de computação tempo-real orientada a objetos. Este trabalho faz parte de um projeto mais amplo, em que, além dos aspectos relativos à implementação tempo-real em plataformas embarcadas, são também analisadas diferentes estratégias de controle preditivo para resolver o problema de controle de trajetória (vide RAFFO (2005)).

1.2 Objetivos

Embora já existam diversos módulos controladores comerciais, a exemplo de CLP's, estes são normalmente destinados a aplicações genéricas e utilizam técnicas de controle clássico. Apesar de tais controladores serem eficazes para tratar problemas de controle convencionais, eles não são apropriados para tratar sistemas mais complexos, de estrutura multivariável e dinâmica de ordem elevada como, por exemplo, o controle de trajetória de veículos autônomos em aplicações de alto desempenho. Assim, o objetivo deste trabalho consiste na análise, projeto e implementação de um sistema tempo-real embarcado com algoritmo de controle preditivo destinado a tratar o problema de seguimento de trajetória de robôs móveis dotados de rodas, dos tipos diferencial e direcional.

A utilização da técnica de controle preditivo baseado em modelo tem como objetivo fazer uso das características favoráveis desta técnica para tratar o problema de seguimento de trajetória, como a possibilidade de utilização de referência futura, de modo a tirar proveito da existência de trajetória pré-determinada na maioria das aplicações de veículos autônomos, bem como a possibilidade de considerar restrições no cálculo da lei de controle, possibilitando o tratamento das restrições intrínsecas dos sistemas eletromecânicos presentes nos robôs móveis.

Na área de sistemas tempo-real, tem-se como objetivo a utilização de conceitos de computação tempo-real orientada a objetos nas etapas de análise, projeto e desenvolvimento do sistema, visando o emprego de uma metodologia bem estruturada em todas as etapas do projeto. A estrutura lógica composta por objetos também pretende contribuir para a manutenção e adaptação do sistema. A modelagem é realizada utilizando a linguagem UML, definida como a linguagem padrão para sistemas orientados a obje-

tos, auxiliada pelo perfil UML-TR, desenvolvido para representar restrições temporais de sistemas tempo-real em diagramas UML. Assim, este trabalho também contribui para a avaliação dos benefícios da utilização do paradigma de orientação a objeto ao longo das etapas de desenvolvimento de sistemas tempo-real complexos, ao passo que analisa diferentes plataformas embarcadas para o uso em controle de robôs e veículos autônomos.

Adicionalmente, este trabalho contribui para a área de robótica móvel realizando a análise, desenvolvimento e implementaçao de sistema tempo-real embarcado para tratar o problema de seguimento de trajetória em aplicações de alto desempenho, analisando de forma detalhada tanto os aspectos referentes à técnica de controle preditivo, quanto os requisitos temporais necessários à implementação em plataforma real dos algoritmos em questão.

1.3 Estrutura da Dissertação

A organização deste volume se dá da seguinte forma:

- O Capítulo 2 descreve questões gerais sobre robôs móveis e veículos autônomos, apresentando as principais motivações para o desenvolvimento e emprego destes em diversas aplicações. São apresentadas as diferentes áreas de estudo que compõem um sistema de navegação autônoma e a forma como elas se relacionam e contribuem para o funcionamento do sistema como um todo.

- O Capítulo 3 descreve o método de controle preditivo baseado em modelo. Esta descrição se inicia com uma introdução do método e em seguida apresenta uma breve perspectiva histórica do CPBM. Também são apresentados os elementos básicos do algoritmo de controle preditivo baseado em modelo e alguns exemplos bibliográficos de aplicações do CPBM a robôs móveis autônomos.

- No Capítulo 4 são apresentados os principais conceitos da computação tempo-real orientada a objetos empregados no desenvolvimento deste trabalho, bem como a metodologia de modelagem utilizada, a linguagem UML e o perfil UML-TR, utilizados para descrever os requisitos necessários ao SCST.

- O Capítulo 5 trata da modelagem de robôs diferenciais e direcionais, bem como da técnica de controle utilizada para tratar o problema de seguimento de trajetória. Na etapa de modelagem dos robôs, é tratada a obtenção dos modelos cinemáticos, sendo apresentados métodos de linearização e de utilização de sistema de coordenadas locais fixo ao robô. Também nesta etapa, é apresentado o modelo dinâmico de robôs direcionais. Em seguida são tratadas estratégias de geração de trajetória de aproximação, apresentando o algoritmo *pure pursuit*, necessário à utilização de modelos linearizados. Na etapa de desenvolvimento de controladores, é apresentado o algoritmo de CPBM denominado *Controle Preditivo Generalizado* (*Generalized Predictive Control*, GPC) e a forma como este é aplicado ao SCST. Utilizando-se o GPC aplicado ao modelo cinemático, são apresentados resultados de simulações computacionais onde se verifica o desempenho da estrutura de controle proposta. Por fim, é analisada a estrutura necessária ao controle em cascata dos comportamentos cinemático e dinâmico, utilizando-se o sistema de controle desenvolvido.

- No Capítulo 6 é apresentada a modelagem e projeto orientado a objetos desenvolvidos para o sistema de controle de seguimento de trajetória, com base na metodologia e nos conceitos apresentados no Capítulo 4. A primeira parte consiste na definição detalhada do problema e na descrição das funcionalidades necessárias ao sistema. Em seguida, são definidas as principais restrições temporais presentes no sistema desenvolvido, juntamente com a realização da modelagem das classes que compõem o sistema, bem como os comportamentos e relacionamentos destas.

- O Capítulo 7 apresenta os detalhes de implementação do sistema em diferentes plataformas destinadas a sistemas embarcados e o resultado de ensaios experimentais utilizando estas plataformas embarcadas em dois robôs móveis, sendo um do tipo diferencial e um do tipo direcional. Também é apresentado um estudo da influência do atraso na execução das tarefas computacionais no desempenho sistema.

- O Capítulo 8 apresenta as conclusões finais sobre o trabalho realizado e as perspectivas futuras de pesquisa e aprimoramento.

- O Apêndice A apresenta a organização das classes do sistema em pacotes, que podem ser utilizados em outras aplicações de modo a promover a reutilização de software.

- O Apêndice B apresenta as características específicas da arquitetura das plataformas DSP e POWERPC utilizadas nos testes de implementação do sistema.

Capítulo 2

Veículos Autônomos

2.1 Introdução

Este capítulo tem a finalidade de apresentar os principais aspectos que envolvem o processo de navegação veicular de forma autônoma e tem objetivo de situar o leitor no contexto em que se desenvolve este trabalho.

O controle de veículos ou robôs autônomos é uma área que vem motivando muito a pesquisa tanto na indústria quanto no meio acadêmico, devido à sua crescente aplicação em diversas áreas de engenharia como, por exemplo, automação de processos de produção industrial, dispositivos militares, sistemas de apoio ao motorista, exploração espacial, operações de alto risco ou em ambientes de difícil acesso e serviços em geral.

Um robô ou veículo é considerado autônomo quando este é apto a realizar navegação automática, ou seja, é capaz de dirigir e reagir ao meio sem controle externo (WIT *et al.*, 2004). Para possibilitar esta navegação automática, diversos campos de pesquisa se desenvolveram, dando origem a técnicas como detecção de postura, visão, planejamento de trajetória, projeto de controle, entre outros (DE WIT *et al.*, 1993), que implicam a necessidade de sistemas computacionais embarcados para realizar tarefas de instrumentação e processamento digital de sinais em tempo-real, além de sensores e atuadores eletro-mecânicos específicos.

Com o avanço das pesquisas e a evolução da eletrônica digital, a tecnologia desenvolvida chegou à indústria automotiva nos anos 90, quando sistemas computacionais embarcados passaram a ser utilizados pelas montadoras para acrescentar novos recursos

aos seus veículos de série. Foi então que se tornaram populares sistemas como injeção eletrônica de combustível, sistema de frenagem assistidos (freios ABS), controle de tração e *airbag* entre outros (LEDUR, 2003).

O processo de automação veicular para navegação pode ser dividido em três etapas: planejamento e geração da trajetória, localização do veículo dentro do ambiente e seguimento da trajetória (WIT *et al.*, 2004; GU e HU, 2002). Os sistemas necessários à execução destas tarefas podem ser divididos em diferentes níveis, conforme a estrutura hierárquica da Figura 2.1. Os sistemas do primeiro nível realizam as tarefas fundamentais do robô ou veículo, como frenagem, aceleração e guiamento, tendo como exemplo os sistemas de injeção eletrônica. No segundo nível, encontram-se os sistemas que atuam sobre o comportamento dinâmico do veículo, como sistemas ABS e ESP. No terceiro nível, estão presentes os controladores responsáveis por guiar o veículo de acordo com uma trajetória pré-determinada. Esta trajetória é definida no quarto e mais alto nível, onde estão os sistemas que realizam o planejamento de trajetória e, caso necessário, a sua adaptação de forma dinâmica ao longo do trajeto, como, por exemplo, o contorno de obstáculos.

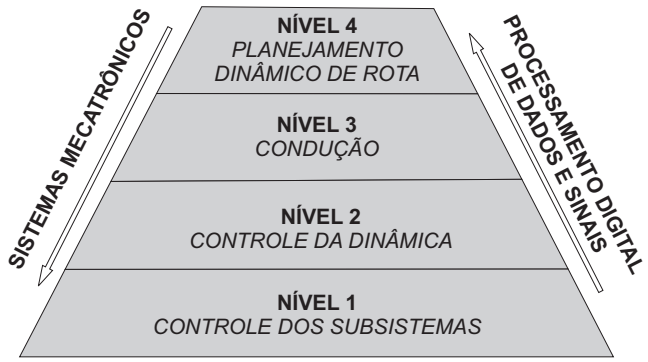

Figura 2.1: Hierarquia dos sistemas de controle de um veículo autônomo (JUNG *et al.*, 2005).

Os sistemas presentes nos níveis mais baixos da hierarquia são caracterizados por controladores de dispositivos eletromecânicos capazes de atuar diretamente sobre a estrutura física do robô ou veículo, servindo de base para a implementação dos níveis superiores. Estes níveis superiores são compostos por sistemas executando tarefas computacionais complexas e por isso exigem dispositivos eletrônicos embarcados com capacidade de processamento em tempo-real. A complexidade de tais sistemas, assim

como suas rigorosas restrições temporais, requerem uma metodologia bem estruturada nas etapas de projeto, desenvolvimento e manutenção, de modo a torná-los coesos, modulares e fáceis de entender, corrigir e modificar.

A seguir serão abordados os principais sistemas necessários ao processo de automação de um veículo autônomo e os tipos de controle e objetivos que se tem no planejamento de movimento para estes veículos.

2.2 Planejamento e Geração de Trajetória

O planejamento de trajetória define um caminho de um ponto a outro englobando uma lista de segmentos de caminho de vários tipos: linhas, arcos, *splines*[1], funções polinomiais, *clothoids*[2], espirais cúbicas, etc (TOUNSI e CORRE, 1996). Já o gerador de trajetória é incumbido de gerar um caminho suave e factível entre a posição atual do veículo e o ponto de destino (GU e HU, 2002), funcionando como um navegador embarcado que continuamente fornece uma referência apropriada para o controlador, uma vez que muitos planejadores de trajetória não consideram as restrições do comportamento dinâmico (MURRAY e SASTRY, 1993). Esta referência, por sua vez, é determinada através do plano de rota, das capacidades de aceleração e desaceleração do veículo e da atual posição deste (NELSON e COX, 1988).

Vários trabalhos foram realizados propondo diferentes técnicas para gerar a trajetória de referência. Em AMIDI (1990) é apresentada a técnica denominada *pure pursuit*, na qual um ponto sobre o caminho desejado é escolhido a partir de uma distância previamente determinada, denominada *look-ahead*, e arcos são calculados unindo a posição atual do veículo a este ponto. Em OLLERO e AMIDI (1991), esta técnica é aprimorada através da adaptação on-line da distância *look-ahead*. Em GÓMEZ-ORTEGA e CAMACHO (1996) a geração de trajetória é realizada com algumas técnicas de controle ótimo, nas quais uma lista de consecutivos pontos de re-

[1]Uma spline é uma curva definida matematicamente por dois ou mais pontos de controle. Os pontos de controle que ficam sobre a curva são chamados de nós.

[2]Uma *clothoid* é uma curva suave cuja curvatura é uma função linear do seu comprimento. A clothoid aparece no problema de encontrar a menor curva que une dois pontos conhecidos, dados o ângulo tangente e a curvatura destes e a derivada da curvatura.

ferência, todos eles localizados sobre o caminho desejado, são escolhidos, em vez de um ponto isolado. Em WIT *et al.* (2004) é apresentada a técnica *vector pursuit*, similar ao *pure pursuit*, baseada na "Teoria dos Helicóides"(BALL, 1900), que gera a trajetória até o ponto de destino considerando também a orientação desejada neste ponto. Em PIAZZI *et al.* (2004) é proposto o método denominado $G^3 - spline$, baseado em um polinômio *spline* de sétima ordem que permite a interpolação de uma seqüência arbitrária de pontos com as associadas orientações arbitrárias, considerando também a velocidade com relação ao comprimento do arco de curvatura (derivada da curvatura) nos pontos arbitrários.

Visando a implementação em tempo-real em plataformas embarcadas, este trabalho utiliza a técnica *pure pursuit* que apresenta uma boa relação entre custo computacional e desempenho, conforme mostraram os resultados obtidos em OLLERO e AMIDI (1991).

2.3 Monitoramento do Meio

Sistemas de controle para navegação autônoma de veículos necessitam de informações precisas sobre a localização do robô e sobre as condições do ambiente no qual ele se encontra, para que possa chegar com segurança a seu ponto de destino ou para rastrear a trajetória de referência. O sensoreamento da localização do veículo e a monitoração de obstáculos no trajeto podem ser feitos usando vários métodos de aquisição de dados, incluindo sistemas de percepção, bem como sensores internos e técnicas de odometria (OLLERO e HEREDIA, 1995).

Os sensores têm a tarefa de enviar ao sistema informações sobre as condições do veículo, a sua localização e como é o meio ao seu redor. Com o objetivo de ter um melhor desempenho, realiza-se a fusão dos dados enviados pelos sensores, tornando assim o sistema de controle mais robusto e confiável (LEDUR, 2003; JUNG *et al.*, 2005). Como pode ser visto na Figura 2.2, os sensores utilizados em automação veicular podem ser divididos em três categorias: de navegação, de reconhecimento de rota e de obstáculos.

No sistema de seguimento de trajetória desenvolvido neste trabalho, são utiliza-

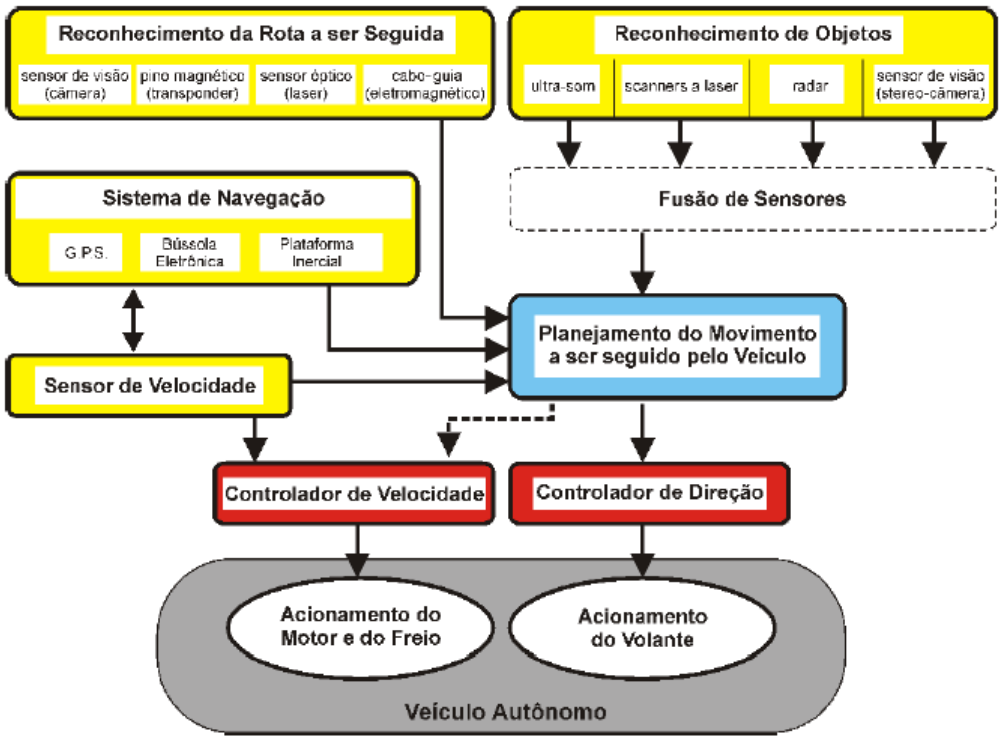

Figura 2.2: Estrutura embarcada para navegação autônoma (JUNG *et al.*, 2005).

dos sensores de navegação e de reconhecimento de obstáculos. São estes a bússola eletrônica, desenvolvida em GOMES *et al.* (2000), e o sensor anti-colisão baseado em ultra-som.

2.4 Seguimento de Trajetórias

Na área de robótica móvel autônoma, uma grande parcela de trabalhos é dedicada ao desenvolvimento de estratégias de controle para seguimento de trajetórias. Este problema se baseia no projeto de técnicas de controle que visam posicionar o veículo sobre um caminho previamente determinado e está intimamente relacionado com a modelagem do robô necessária para o projeto da lei de controle, sendo que alguns modelos são definidos como sistemas não-holonômicos, o que, conforme descrito no Capítulo 1, impõe muitas dificuldades no projeto de controladores.

Os projetos de leis de controle para estes robôs têm sido desenvolvidos seguindo três metodologias (DE WIT *et al.*, 1993):

- *Estabilização em um Ponto*: Para um sistema linear invariante no tempo, se

os autovalores instáveis são controláveis, um ponto de equilíbrio pode ser assintoticamente estabilizado por uma realimentação estática suave e invariante no tempo. Entretanto, para sistemas não-lineares e com restrições não-holonômicas, isto não é possível. Assim, técnicas lineares antes utilizadas não podem mais ser consideradas. Neste caso, usualmente, leis de controle variantes no tempo ou não suaves são utilizadas a fim de transpor as restrições não-holonômicas (KÜHNE, 2005);

- *Rastreamento de Trajetória*: Devido às limitações impostas pelas restrições não-holonômicas, métodos de controle foram desenvolvidos abandonando a idéia de estabilização em um ponto e procurando obter convergência para uma trajetória. Assim, este problema é realizado em duas etapas: primeiro uma trajetória é calculada *off-line* considerando tempo fixo e, em seguida, uma lei de controle é projetada a fim de fazer com que o robô siga a trajetória calculada previamente (KÜHNE, 2005). Assume-se, aqui, que a velocidade tangencial é uma entrada de controle do modelo cinemático.

- *Seguimento de Trajetória*: Este caso é bastante semelhante ao caso acima, pois tem também como objetivo que o veículo siga uma trajetória previamente calculada. Porém, esta é definida de forma geométrica e, normalmente, a restrição em relação ao tempo não existe. Portanto, geralmente se considera que a velocidade tangencial é mantida constante e a convergência é obtida apenas através da velocidade angular (KÜHNE, 2005).

Muitas destas pesquisas usam apenas modelos cinemáticos em coordenadas cartesianas para completar estas tarefas, tendo como argumentos baixas velocidades, baixas acelerações e não ocorrência do deslizamento das rodas. Assim, os modelos cinemáticos são válidos (RAFFO, 2005). Porém, o modelo cinemático não é adequado para representar com exatidão o comportamento de um veículo autônomo quando as condições acima não são obedecidas, de modo que, nestes casos, o modelo que descreve o comportamento dinâmico deve ser considerado. Os modelos cinemáticos descrevem o comportamento do veículo em função da velocidade e orientação das rodas, enquanto os modelos dinâmicos descrevem o comportamento em função das forças generalizadas

aplicadas pelos atuadores e das forças externas (LAGES, 1998).

Os sistemas para controle de seguimento de trajetória podem então ser classificados em três categorias: o controle considerando apenas o modelo cinemático, o controle considerando apenas o modelo dinâmico e o controle utilizando tanto o modelo cinemático quanto o dinâmico. Neste trabalho, conforme mencionado anteriormente, é realizado o projeto e a implementação de um sistema tempo-real embarcado capaz de ser aplicado tanto ao modelo cinemático quanto ao modelo dinâmico de robôs dos tipos diferencial e direcional utilizando algoritmo de CPBM linear.

2.5 Conclusões

Este capítulo apresentou a definição, os campos de aplicação e as principais características de um robô ou veículo autônomo. Foram elencados os principais sistemas necessários à navegação autônoma, a forma como estes se organizam em diferentes níveis e como se relacionam.

O problema de seguimento de trajetórias em robôs móveis foi brevemente discutido, sendo introduzida a estratégia desenvolvida no Capítulo 4 com base na aplicação da técnica de CPBM abordada no Capítulo 3.

A análise das características de robôs autônomos e do problema de seguimento de trajetória também deixou explícita a necessidade de processamento em tempo-real em sistemas embarcados para estas aplicações. Isso remete à necessidade de utilização de metodologia específica e bem definida durante todas as etapas da concepção do sistema de controle, juntamente com a definição detalhada de todas as restrições temporais.

Capítulo 3

Controle Preditivo Baseado em Modelo

3.1 Introdução

O termo Controle Preditivo Baseado em Modelo, ou CPBM, representa um amplo grupo de métodos de controle que utilizam explicitamente um modelo do processo a ser controlado para estimar o seu comportamento futuro e, com isso, obter a ação de controle através da minimização de uma função custo. A técnica de controle preditivo surgiu nos anos 70 e tem se desenvolvido consideravelmente desde então. Esta filosofia procura refletir a habilidade humana de tomar decisões e praticar ações que produzam resultados desejados ao longo de um intervalo de tempo à frente. O homem seleciona estas ações e decisões com base no conhecimento (modelo) do sistema em questão, e atualiza constantemente suas decisões à medida em que novas observações são realizadas. De forma similar, o controle preditivo dimensiona as ações de controle a partir de predições baseadas nas informações disponíveis (medições) e no modelo do processo. Embora hajam muitas estratégias de controle preditivo, todas apresentam, em maior ou menor grau, as seguintes idéias básicas:

- Uso explícito de um modelo para realizar a predição das saídas do processo em um intervalo de tempo futuro.
- Cálculo da seqüência de controles futuros de modo a minimizar uma função objetivo.
- Estratégia de horizonte deslizante, na qual, a cada amostra, o horizonte de

predição é deslocado em direção ao futuro. Com isso, para cada seqüência de controles futuros calculada em cada amostra, apenas o primeiro sinal é aplicado ao sistema.

Todos os algoritmos propostos com a técnica de controle preditivo, e que seguem estas idéias, diferem apenas uns dos outros no modelo utilizado para representar o processo e os ruídos, e na função custo a ser minimizada. Porém, uma vez compreendida a filosofia do CPBM, os detalhes de implementação de cada técnica se tornam mais fáceis de ser analisados.

As principais vantagens que podem ser atribuídas ao CPBM sobre a maioria dos métodos de controle são:

- É particularmente atrativo para operadores com pouco conhecimento técnico, pois utiliza conceitos intuitivos e o ajuste é relativamente fácil;
- Pode ser utilizado para controlar uma grande variedade de processos, desde aqueles que apresentam dinâmica simples até os mais complexos, tratando, inclusive, de sistemas com atraso de transporte;
- O caso multivariável é tratado com facilidade;
- Possui compensação intrínseca para os casos de atraso de transporte;
- Introduz o controle *feed forward* de um modo natural para compensar perturbações mensuráveis;
- O controlador resultante é uma lei de controle linear fácil de implementar;
- A extensão para a tratamento de restrições é conceitualmente simples, sendo que estas podem ser sistematicamente incluídas durante a etapa de projeto;
- É muito útil quando a referência futura é conhecida, como em aplicações na área de robótica, por exemplo;
- É uma metodologia totalmente aberta, baseada em um conjunto de princípios, que permite muitas extensões em trabalhos futuros.

Uma das maiores razões para o grande êxito desta técnica de controle em aplicações industriais pode ser atribuída ao fato de que o CPBM talvez seja a abordagem mais geral para tratar o problema de controle de processos no domínio do tempo. Além disso, incorpora características de controle ótimo, controle estocástico e tratamento de restrições, entre outras, podendo ser aplicado na maioria dos processos não-lineares comumente encontrados na indústria, bem como aplicações que vão desde manipuladores até anestesia clínica. Os bons resultados obtidos em aplicações variadas demonstram a capacidade do CPBM em atingir níveis de controle altamente eficientes e estáveis, capazes de operar por grandes períodos de tempo sem necessitar de intervenção (CAMACHO e BORDONS, 1998).

3.2 Perspectiva Histórica

Os primeiros trabalhos sobre controle preditivo surgiram no final dos anos 70 com o surgimento de vários artigos demonstrando interesse industrial no CPBM, com destaque para (RICHALET *et al.*, 1976), apresentando o Controle Algorítmico Baseado em Modelo (*Model Algorithm Control*, MAC), e para (CUTLER e RAMAKER, 1988), apresentando o Controle de Matriz Dinâmica (*Dinamic Matrix Control*, DMC). Ambos os algoritmos eram caracterizados pelo uso explícito de um modelo para predizer o efeito de ações futuras de controle nas saídas do processo. Enquanto o MAC utilizava como modelo a resposta ao degrau do sistema, o DMC utiliza a resposta ao impulso.

O controle preditivo teve de imediato muito sucesso, principalmente em aplicações na indústria química, devido à simplicidade do algoritmo e o uso de resposta ao impulso ou resposta ao degrau como modelo do sistema em vez de formulações matemáticas mais complexas como, por exemplo, modelos no espaço de estados. A resposta de sistemas frente a excitações do tipo impulso ou degrau é facilmente obtida em sistemas SISO e MIMO nas plantas industriais e dispensa conhecimento aprofundado de modelagem de sistemas. Mas apesar do sucesso obtido até então, praticamente não havia estudos formais apresentando análises teóricas de estabilidade e robustez para estes algoritmos.

Uma outra linha de trabalhos na área de CPBM surgiu de forma independente no meio acadêmico em torno de idéias relacionadas ao controle adaptativo e apresentando

características diferenciadas. Nesta linha de trabalhos estão incluídos o "controle preditivo generalizado" (*Generalized Predictive Controller*, GPC) (CLARKE *et al.*, 1987), o "controle adaptativo de predição estendida" (*Extended Prediction Self Adaptive Control*, EPSAC) (KEYSER e CUAWENBERGHE, 1985), o "controle adaptativo de horizonte estendido" (*Extended Horizon Adaptive Control*, EHAC) (YDSTIE, 1984) e o "controle preditivo unificado" (*Unified Predictive Control*, UPC) (SOETERBOEK, 1992), que modelam o processo através de função de transferência e as perturbações através de um modelo autorregressivo integrado de média móvel (*Autorgeressive Integrated Moving Average*, ARIMA) (GOODWIN e SIN, 1984), enquanto que as predições da saída do processo são obtidas por meio de preditores ótimos. Algumas destas técnicas foram capazes de provar estabilidade por meio de alguns teoremas através da busca de relações com o espaço de estados, e também apresentaram estudos da influência de filtros polinomiais no aumento da robustez. Porém, em algumas situações, o caso de horizonte finito se demonstrava uma barreira que impedia análises mais genéricas de estabilidade e robustez.

Assim, a partir desta necessidade, uma nova linha de trabalhos em controle preditivo com garantia de estabilidade surgiu nos anos 90. Os métodos CRHPC, SIORHC e *stable* GPC provaram ser estáveis através da imposição de restrições às saídas após um horizonte finito. Atualmente, os trabalhos na área de CPBM continuam se diversificando, e algumas questões como identificação de modelo, predição e estimação de perturbações não mensuráveis, tratamento sistemático de incertezas e CPBM não-linear despontam como os principais desafios a serem confrontados nos trabalhos futuros.

3.3 Elementos do CPBM

A técnica de CPBM se baseia em três elementos fundamentais, que estão presentes em todos os algoritmos propostos:

- o modelo de predição;
- a função objetivo;
- o método para a obtenção da lei de controle.

As diferentes opções escolhidas para cada um destes elementos caracterizam os diferentes algoritmos propostos na literatura (NORMEY-RICO, 2003).

3.3.1 O Modelo de Predição

O modelo de predição consiste no elemento mais importante dos controladores preditivos, devendo ser completo o suficiente para capturar toda a dinâmica do processo, além de possibilitar o cálculo das predições e ao mesmo tempo ser intuitivo e permitir análises teóricas (CAMACHO e BORDONS, 1998). Em geral, o modelo de predição é separado em duas partes. A primeira consiste no modelo do processo, que representa a relação entre as saídas e as entradas mensuráveis, que podem ser variáveis manipuláveis e perturbações mensuráveis. A segunda parte consiste no modelo das perturbações, que descreve o comportamento do sistema devido a perturbações não mensuráveis, ruídos e erros de modelagem.

O Modelo do Processo

Nas diversas formulações de CPBM são encontradas praticamente todas as formas de se modelar um processo. Dentre as formas de modelagem, as mais utilizadas são:

- Resposta Impulsiva. É utilizada no MAC e em casos especiais no GPC e no EPSAC. A relação entrada saída é dada por:

$$y(k) = \sum_{i=1}^{\infty} h_i u(k-i)$$

onde h_i são as amostras da reação de saída do processo em resposta à aplicação de um impulso na entrada do mesmo. Em geral, como esta seqüência é infinita, a resposta é truncada nos primeiros N valores, o que restringe a utilização deste modelo a plantas estáveis, onde $h_i \to 0$ quando $i \to \infty$:

$$y(k) = \sum_{i=1}^{N} h_i u(k-i) = H(z^{-1})u(k)\ , \tag{3.1}$$

onde $H(z^{-1}) = h_1 z^{-1} + h_2 z^{-2} + \cdots + h_N z^{-N}$, e z^{-1} é o operador atraso unitário.

A predição da saída em $k+j$ calculada no instante k, ($\hat{y}(k+j|k)$), usando este modelo é calculada como:

$$\hat{y}(k+j|k) = \sum_{i=1}^{N} h_i u(k+j-i|t) = H(z^{-1})u(k+j|k) \ .$$

Este método é bastante utilizado na prática devido às vantagens que oferece, como: (a) é intuitivo; (b) não precisa de conhecimento *a priori* do processo e pode ser usado em plantas multivariáveis sem acrescentar complexidade e (c) descreve, de maneira simples, efeitos mais complexos da dinâmica do processo, como atrasos e comportamentos de fase não mínima.

Por outro lado apresenta alguns inconvenientes: (a) não pode ser usado com plantas instáveis e (b) necessita utilizar um grande número de parâmetros para descrever o modelo. Geralmente N pode ser 40 ou 50, valor que pode aumentar ainda mais se o processo tiver um atraso grande.

- Resposta ao degrau. É usado pelo DMC e suas variantes. É similar ao anterior, mas usa um degrau unitário como sinal de entrada. Para sistemas estáveis a resposta truncada é:

$$y(k) = y_0 + \sum_{i=1}^{N} g_i \triangle u(k-i) = y_0 + G(z^{-1})(1-z^{-1})u(k) \ , \tag{3.2}$$

onde $\Delta u(k) = u(k) - u(k-1)$ e os g_i são as amostras da saída obtida ao aplicar o degrau. Considerando o sistema no ponto de operação y_0 a predição pode ser calculada como:

$$\hat{y}(k+j|k) = \sum_{i=1}^{N} g_i \Delta u(k+j-i|k) \ .$$

A relação entre este modelo e o de resposta impulsiva é dada por:

$$h_i = g_i - g_{i-1} \qquad g_i = \sum_{p=1}^{i} h_p \ .$$

Resulta claro que este modelo tem as mesmas vantagens e inconvenientes que o anterior.

- Função de Transferência. Este modelo é usado no GPC, UPC, EPSAC, EHAC entre outros. Utiliza o conceito de função de transferência $G = B/A$:

$$A(z^{-1})y(k) = B(z^{-1})u(k) \ ,$$

com

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{na} z^{-na} \\ B(z^{-1}) &= b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{nb} z^{-nb} \ . \end{aligned}$$

A predição é calculada como:

$$\hat{y}(k+j|k) = \frac{B(z^{-1})}{A(z^{-1})} u(k+j|k) \ .$$

Esta representação tem como principais vantagens poder ser usada para plantas instáveis e precisar, em geral, de poucos parâmetros para descrever o comportamento do sistema (o atraso, por exemplo, pode ser descrito apenas com um parâmetro). Já seu principal inconveniente é a necessidade de conhecer *a priori* a ordem dos polinômios A e B quando o modelo deve ser identificado a partir de dados experimentais.

- Espaço de estados. É usado no controle preditivo funcional PFC RICHALET *et al.* (1987) e tem a seguinte representação:

$$\begin{aligned} x(k) &= P_x x(k-1) + H_x u(k-1) \\ y(k) &= Px(k) \end{aligned}$$

onde x é o estado e P_x, H_x e P são matrizes de dimensões compatíveis. A predição é calculada como:

$$\hat{y}(k+j|k) = P\hat{x}(k+j|k) = P[P_x^j x(k) + \sum_{i=1}^{j} P_x^{i-1} H_x u(k+j-i|k)] \ ,$$

Sua principal vantagem é que pode ser usada diretamente para processos multivariáveis. Como inconvenientes cabe mencionar que em geral os estados não tem significado físico e que na maioria das vezes é necessário o uso de observadores, aumentando assim a complexidade de cálculo do controle.

- Outros modelos. Todos os modelos anteriormente descritos consideram somente o comportamento linear ou linearizado do processo, e, por sua simplicidade, são os mais utilizados. Porém, modelos não lineares também podem ser usados para descrever a dinâmica do processo quando os modelos lineares não geram bons resultados. É importante salientar, entretanto, que a solução do problema nestes casos apresenta alguns inconvenientes adicionais como: (a) a obtenção do modelo do processo e (b) a complexidade dos algoritmos de controle resultantes. Redes neurais (TAN e KEYSER, 1994) ou lógica nebulosa (SKRJANC e MATKO, 1994) podem ser usadas em algumas aplicações para determinar o modelo de predição.

  O CPBM não linear é um campo aberto para pesquisas tanto na área de determinação de modelos como nos procedimentos e algoritmos de otimização para o cálculo do controle.

Modelo das Perturbações

Tão importante quanto o modelo do processo é o modelo das perturbações. O modelo mais utilizado para a descrição de perturbações determinísticas e estocásticas é o conhecido como modelo autoregressivo integrado de média móvel (Auto-Regressive and Integrated Moving Average (ARIMA)). Através do modelo ARIMA, as diferenças entre a saída do modelo e do processo são modeladas por:

$$n(k) = \frac{C(z^{-1})e(k)}{D(z^{-1})}$$

onde o polinômio $D(z^{-1})$ inclui explicitamente um integrador $\Delta = 1 - z^{-1}$, e $e(k)$ é um ruído branco de média zero. Os demais parâmetros dos polinômios C e D são usados para descrever as características estocásticas de n. Este modelo permite representar mudanças aleatórias, *off-sets* e outros fenômenos normalmente encontrados nos meios industriais, sendo usado diretamente no GPC, EPSAC, EHAC e UPC e, com algumas modificações, em outros controladores.

Alguns casos particulares importantes são: (a) o modelo usado em DMC,

$$n(k) = \frac{e(k)}{1 - z^{-1}} \ ,$$

onde a melhor predição para $\hat{n}(k+j|k)$ é $n(k)$, tendo em vista que $e(k)$ tem média zero; ou (b) o modelo usado no PFC

$$n(k) = \frac{e(k)}{(1 - z^{-1})^2} \ ,$$

onde $\hat{n}(k+j|k) = n(k) + (n(k) - n(k-1))k$.

Outras variações destes modelos e estudos sobre o efeito do modelo das perturbações no sistema de controle podem ser encontradas em (BERGH e MACGREGOR, 1987; PALMOR, 1982).

Resposta livre e forçada

Uma característica comum nos CPBM, é a utilização dos conceitos de resposta livre e forçada. A idéia é considerar à seqüência de controle composta por duas partes:

$$u(k) = u_f(k) + u_c(k)$$

- $u_f(k)$ correspondente aos valores passados da entrada e que são mantidos iguais aos valores da variável manipulada,

$$\begin{aligned} u_f(k-j) &= u(k-j) \text{ para } j = 1, 2, \cdots \\ u_f(k+j) &= u(k-1) \text{ para } j = 0, 1, 2, \cdots \end{aligned}$$

- $u_c(k)$ é zero no passado e igual aos controles a serem aplicados no futuro:

$$\begin{aligned} u_c(k-j) &= 0 \text{ para } j = 1, 2, \cdots \\ u_c(k+j) &= u(k+j) - u(k-1) \text{ para } j = 0, 1, 2, \cdots \end{aligned}$$

Desta forma, a predição da saída do processo pode ser separada em duas partes: a resposta livre $y_f(k)$, que corresponde à predição quando a entrada é igualada a $u_f(k)$; e a resposta forçada $y_c(k)$, que corresponde às predições quando o controle é igual a $u_c(k)$. Esta idéia pode ser usada para uma implementação bastante simples e intuitiva dos algoritmos de CPBM.

3.3.2 Função Objetivo

Em geral, os diversos algoritmos de CPBM utilizam diferentes funções de custo para calcular a lei de controle, porém todos eles têm como objetivo minimizar o erro entre a saída futura (y) e a referência desejada (y_{ref}) penalizando o esforço de controle Δu.

Assim, a expressão mais geral desta função objetivo é:

$$J = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(k+j|k) - y_{ref}(k+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\triangle u(k+j-1)]^2 \qquad (3.3)$$

Os elementos desta função são:

- Parâmetros: N_1, N_2, N_u, $\delta(k)$ e $\lambda(k)$. N_1 e N_2 são os horizontes de predição mínimo e máximo, N_u é o horizonte de controle. Os valores destes índices têm uma interpretação clara, já que definem os instantes em que se deseja que a referência siga a saída e quando é importante limitar a ação de controle. Assim, se, por exemplo, N_1 é grande, isso implica que não é importante o erro cometido nos primeiros $N_1 - 1$ instantes e a resposta obtida tenderá a ser suave. No caso particular de sistemas com um atraso de valor d, é lógico escolher $N_1 > d$, já que não haverá resposta do sistema à entrada $u(k)$ até o tempo $k = d$. Variando N_u é possível penalizar, durante mais ou menos tempo, a ação de controle. Os coeficientes $\delta(j)$ e $\lambda(j)$ são as seqüências de ponderação do erro e do esforço de controle e geralmente são escolhidas constantes ou exponenciais ao longo do horizonte. Por exemplo, uma função do tipo:

$$\delta(j) = \alpha^{N_2 - j}$$

  permite variar a penalização do erro em diferentes partes do horizonte. Assim, por exemplo, para gerar respostas mais suaves, escolhe-se um de valor α entre 0 e 1 de forma tal que sejam mais penalizados os últimos valores do erro dentro do horizonte.

- Trajetória de referência:

  Uma das vantagens do CPBM é a possibilidade de utilizar o conhecimento dos valores futuros da referência, quando disponíveis, para o cálculo do sinal de controle, o que permite, por exemplo, que o sistema atinja mais rapidamente o novo valor desejado. Esta característica se torna interessante em algumas aplicações como em robótica móvel e manipuladora, em servoacionamentos e em processos do tipo batelada, nos quais as referências futuras são conhecidas *a priori*.

  Os valores de $y_{ref}(k + j)$ utilizados na função objetivo não são necessariamente coincidentes com a referência real do sistema. Normalmente, nas aplicações

práticas, utilizam-se estratégias para suavizar as mudanças de referência, de forma similar aos filtros utilizados nas estruturas clássicas de controle com dois graus de liberdade. Uma forma típica para esta lei é:

$$y_{ref}(k) = r(t) \quad y_{ref}(k+j) = \alpha y_{ref}(k+j-1)+(1-\alpha)r(k+j) \quad j = 1 \ldots N \quad (3.4)$$

onde α é um parâmetro entre 0 e 1. Esta lei representa um filtro-passa-baixa de primeira ordem que pode ser ajustado para suavizar mais, α próximo de um, ou menos, α próximo de zero, a forma da resposta. Estas idéias são usadas no GPC e no EPSAC para especificar o comportamento desejado para a malha fechada CLARKE e MOTHADI (1989).

- Restrições: Na prática, todos os processos estão sujeitos a restrições tanto nas variáveis de saída como de entrada. Exemplos disto são: os limites máximos e mínimos impostos aos atuadores (ex.: válvulas), a máxima velocidade de variação de um acionamento (ex.: servo-acionamentos), os valores limites que podem ser atingidos pelas saídas de um sistema devido a questões de segurança, etc. Além disso, existem restrições de natureza econômica para o funcionamento do sistema que em geral levam a escolher pontos de operação muito próximos destes limites. Assim, se o controle é corretamente calculado para trabalhar muito próximo daquele ótimo, a qualidade e a relação custo-benefício do processo produtivo são otimizadas (CAMACHO e BORDONS, 1998). Por estes motivos, a inclusão das restrições na função objetivo que se deseja minimizar é importante. Neste sentido, todos os algoritmos de CPBM permitem incluí-las no momento da obtenção do mínimo de J considerando um conjunto de equações do tipo:

$$\begin{aligned} u_{min} \leq \quad & u(k) & \leq u_{max} \quad & \forall k \\ du_{min} \leq \quad & u(k) - u(k-1) & \leq du_{max} \quad & \forall k \\ y_{min} \leq \quad & y(k) & \leq y_{max} \quad & \forall k \end{aligned}$$

Deve ser mencionado aqui que, nestes casos, a solução do mínimo de J não pode ser obtida analiticamente e requer uma carga de cálculo muito maior que no caso sem restrições. Apesar da complexidade de cálculo, a capacidade do CPBM de levar as restrições em consideração é um dos principais motivos do seu sucesso nas aplicações industriais.

O tratamento do CPBM com restrições apresenta diversas dificuldades, tanto teóricas como de implementação. A formulação do problema consiste no correto equacionamento das restrições e num tratamento posterior que é conhecido como "estudo de factibilidade e gestão de restrições". Este tratamento permite o correto funcionamento do algoritmo de otimização, liberando ou suavizando, quando possível, as restrições. Por outro lado, do ponto de vista da implementação do algoritmo de otimização, as pesquisas estão orientadas à melhoria da eficiência e à minimização dos tempos de cálculo. Finalmente, os problemas de estabilidade destes sistemas de controle somente têm sido resolvidos parcialmente e numerosas pesquisas vêm sendo realizadas nos últimos anos (CAMACHO e BORDONS, 1998).

3.3.3 Obtenção da Lei de Controle

Em todos os algoritmos de CPBM, o objetivo é calcular $u(k+j|k)$ para minimizar J. Para isso, é necessário calcular as predições $\hat{y}(k+j|k)$ como função do controle futuro e, a partir do método utilizado por cada algoritmo, substituí-las na função J. No caso de utilizar um modelo linear e sem restrições é possível obter uma solução analítica do mínimo de J. Em outro caso a solução é obtida de forma iterativa por algum método de otimização.

Independente do método utilizado, a solução é, em geral, complexa devido ao número de variáveis envolvidas, principalmente quando os horizontes são grandes. Para reduzir os graus de liberdade deste problema alguns algoritmos propõem estruturar a lei de controle. Isto pode ser feito como no DMC, GPC, EPSAC e EHAC, ajustando o horizonte de controle, N_u, o que implica zerar as variações do controle após um certo valor no horizonte $N_u < N_2$:

$$\Delta u(t+j-1) = 0 \qquad j > N_u$$

Outra forma de estruturar o controle, que é usada no PFC, consiste em calcular o controle como uma combinação de funções pré-estabelecidas:

$$u(k+j) = \sum_{i=1}^{n} \mu_i(k) B_i(j) \tag{3.5}$$

onde os B_i são escolhidos de acordo com o tipo de processo e de referência.

Neste ponto também existe um grande campo para pesquisas em CPBM, já que os problemas de otimização associados ao cálculo do controle ótimo não têm sido resolvidos de forma geral (NORMEY-RICO, 2003).

3.4 Controle Preditivo de Veículo Autônomos

O controle de veículos autônomos utilizando CPBM não é muito freqüentemente encontrado na literatura. A seguir serão citados alguns destes trabalhos.

Em OLLERO e AMIDI (1991), o GPC é aplicado ao problema de seguimento de trajetória do veículo CMU NavLab, uma van comercial. O controle é realizado através do ângulo de orientação do veículo em torno do eixo z, considerando que a velocidade tangencial permanece constante. A função custo envolve o erro de posição e orientação em coordenadas locais além da penalização dos incrementos de controle e a consideração de restrições. A utilização de um modelo linear implica a necessidade de geração de trajetórias de aproximação, de modo a evitar variações muito altas da orientação do veículo.

Em GÓMEZ-ORTEGA e CAMACHO (1996), algoritmos genéticos são utilizados para a otimização não-linear, a fim de diminuir o esforço computacional e tornando possível assim a aplicação em tempo-real. O problema de seguimento de trajetória é solucionado para um veículo autônomo com modelo não-linear e acionamento diferencial. A função custo a ser minimizada inclui um termo que penaliza a proximidade entre o veículo e obstáculos fixos no meio. O caminho é previamente definido considerando obstáculos conhecidos, e o problema de seguimento de trajetória é resolvido com obstáculos não planejados presentes no meio.

Em YANG *et al.* (1998) um controle preditivo inteligente é apresentado, em que um modelo cinemático em redes neurais é utilizado para a predição das saídas, e as entradas de controle são as velocidades tangencial e angular. O veículo autônomo considera seu modelo similar a de um automóvel, com duas rodas traseiras para tração e duas frontais para direção. A função custo envolve o erro de posição e esforço de controle.

Em NORMEY-RICO *et al.* (1999), são comentadas algumas vantagens da utilização do CPBM para o problema de seguimento de trajetória de robôs móveis. Foi utilizado o algoritmo GPC com a predição das saídas realizada pela estrutura do *Preditor de Smith*, SP, sem considerar restrições para o seguimento do caminho. Foi utilizado um modelo em coordenadas locais do robô e a velocidade tangencial foi considerada constante. Utilizou-se a trajetória de aproximação *pure pursuit*, com intuito de evitar grandes variações da orientação do veículo. O algoritmo foi aplicado a um robô com acionamento diferencial.

Em KÜHNE (2005), desenvolve-se um algoritmo não-linear de CPBM em espaço de estados utilizando coordenadas polares aplicado a um robô de acionamento diferencial para estabilização em um ponto e rastreamento de trajetória. O método utilizado é comparado com leis de controle variantes no tempo e descontínuas, mostrando a eficiência do CPBM. É realizado também o controle preditivo linear no espaço de estados utilizando o modelo cinemático do erro, o qual é obtido através de linearizações sucessivas ao longo do horizonte de predição.

Em alguns trabalhos citados acima, são utilizados o modelo cinemático do veículo em coordenadas locais e o modelo cinemático do erro, permitindo o uso de algoritmos de otimização convexa e diminuindo drasticamente o custo computacional necessário (NORMEY-RICO *et al.*, 1999; KÜHNE, 2005). Quando utilizado o modelo em coordenadas locais, se faz necessário o uso de trajetórias de aproximação, pois este é válido somente para pequenas variações do incremento da orientação do veículo. Quando utilizado o modelo cinemático do erro, a cada amostra, são realizadas linearizações sucessivas até um horizonte finito.

Em GÓMEZ-ORTEGA e CAMACHO (1996); NORMEY-RICO *et al.* (1999); KÜHNE (2005) a dinâmica dos veículos é desprezada, pois a velocidade com que estes

executam os trajetos é considerada pequena e a massa dos robôs móveis não influencia o objetivo de seguimento da trajetória. Porém, como citado anteriormente, quando se deseja obter desempenho em velocidades elevadas, forças externas ao veículo devem ser consideradas, de modo que a dinâmica do veículo deve ser considerada pelo sistema de controle.

3.5 Conclusões

Neste capítulo, foi apresentada a técnica de controle preditivo baseado em modelo, sua evolução cronológica, seus elementos fundamentais, as principais vantagens e os principais algoritmos existentes na literatura. Observou-se que, de modo a refletir o comportamento humano, o CPBM utiliza o conhecimento (modelo) de um sistema para prever (predizer) o seu comportamento e, a partir daí, agir sobre este da maneira mais adequada de modo a produzir o efeito desejado. A partir daí surge o modelo do sistema como elemento fundamental do CPBM para a realização correta das predições. Este modelo deve ser preciso o suficiente para realizar predições confiáveis e possibilitar a utilização de horizontes de predição significativos. A utilização de uma função custo permite um seleção criteriosa da ação de controle a ser aplicada, dando margem à utilização de restrições e diferentes estratégias de otimização. A utilização explícita de referência futura coloca o CPBM em um formato interessante para o controle de veículos autônomos. A utilização do CPBM para controle de veículos é detalhada no próximo capítulo.

Capítulo 4

Controle Preditivo Aplicado a Robótica Móvel

4.1 Introdução

A arquitetura de controle de um robô móvel autônomo é dividida em diferentes níveis, como mostra a Figura 2.1. O nível de planejamento dinâmico de trajetória, tratado superficialmente neste trabalho, consiste em um sistema supervisório incumbido de realizar o planejamento da rota a ser seguida, bem como monitorar e evitar obstáculos. O nível de controle dos subsistemas do veículo já se encontra muito difundido em veículos comerciais e permite aos níveis superiores atuarem nos sistemas de direção, aceleração e frenagem por meio de comandos eletrônicos. O foco deste trabalho consiste no desenvolvimento de um controlador preditivo multivariável em tempo-real (TR) para ser aplicado nos níveis de controle de dinâmica e condução do veículo, a partir dos dados fornecidos pelo sistema de planejamento dinâmico de trajetória e fazendo uso das malhas de controle dos subsistemas do veículo. A nível de planejamento dinâmico de trajetória, o trabalho também trata da monitoração da distância do veículo em relação ao ponto de destino sobre o caminho de referência, de modo a gerar trajetórias responsáveis por aproximar o robô a este ponto de destino.

Neste capítulo, será apresentada a modelagem de robôs móveis direcionais e diferenciais, a descrição da técnica de CPBM aplicada ao problema de controle de seguimento de trajetória para estas duas classes de veículos, bem como a estratégia de geração de trajetórias de aproximação. A análise da estrutura dos robôs será direcionada no sentido da obtenção de modelos linearizados, pois trabalhos prévios de CPBM aplicado

ao seguimento de trajetórias de veículos autônomos têm mostrado que o custo computacional associado ao algoritmo de CPBM não-linear o tornam inviável para utilização em sistemas embarcados em TR (KÜHNE, 2005).

A definição do algoritmo de controle linear a ser implementado foi realizada com base no trabalho realizado por RAFFO (2005), onde são comparadas duas abordagens para utilização de CPBM linear aplicado ao problema de seguimento de trajetória. Uma abordagem consiste na utilização de um modelo cinemático do erro, onde se considera a existência de um robô de referência *virtual* sobre a trajetória a ser seguida, descrito pelo mesmo modelo cinemático do robô real. Para isso, considera-se que a trajetória de referência é definida *off-line* e é variante no tempo. O objetivo de usar essa estratégia, para entradas de referência não-nulas, é calcular uma lei de controle linear que faça com que o erro entre o robô e a referência seja nulo (NELSON e COX, 1988; KÜHNE, 2005). A outra abordagem considera o modelo cinemático em coordenadas locais e assume que os incrementos do ângulo de orientação θ do veículo são pequenos a cada amostra. A partir dessas suposições, chega-se ao modelo cinemático linearizado para o veículo. Entretanto, se o veículo não estiver posicionado sobre a trajetória de referência, grandes variações em θ tendem a ocorrer, o que acaba por invalidar o modelo linearizado. Isso remete à necessidade de geração de rotas de aproximação, de modo que o veículo sempre esteja localizado sobre a trajetória de referência (NORMEY-RICO *et al.*, 1999).

Como resultado da comparação entre as duas abordagens, tem-se que o algoritmo CPBM utilizando espaço de estados aplicado ao modelo cinemático do erro é extremamente mais custoso computacionalmente se comparado ao algoritmo GPC aplicado ao modelo cinemático em coordenadas locais. Isto se dá pela necessidade do primeiro algoritmo de linearizar sucessivamente o modelo do veículo a cada passo amostral para todo o horizonte de predição (RAFFO, 2005). Assim, para realizar o controle de seguimento de trajetória, foi utilizado o algoritmo GPC, pois além de apresentar baixo custo computacional, pode ser aplicado diretamente aos modelos cinemáticos e dinâmicos a serem obtidos para as duas classes de veículos.

4.2 Modelagem de um Robô Diferencial

Neste trabalho, é considerado robô diferencial um robô de corpo rígido composto por duas rodas não deformáveis alinhadas ao veículo e de acionamento independente, que realiza curvas através da diferença de velocidade entre estas duas rodas. A análise da estrutura do veículo será direcionada no sentido da obtenção de um modelo cinemático linearizado a partir do modelo cinemático não-linear de robôs móveis. Devido ao fato de a estrutura de acionamento diferencial estar pouco presente em veículos de grande porte, os aspectos dinâmicos referentes a esta estrutura foram desconsiderados. A Figura 4.1 apresenta a estrutura do robô diferencial.

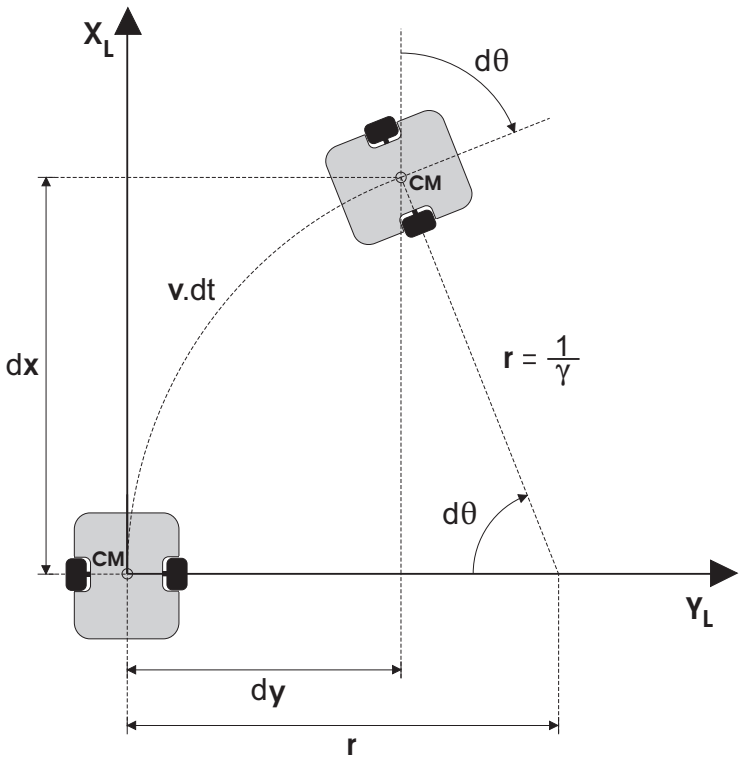

Figura 4.1: Configuração de um robô diferencial.

O ponto de guiamento do robô, ou seja, o ponto sobre o veículo que é designado a seguir a trajetória de referência (NELSON, 1989) é definido no ponto entre as duas rodas, assim como o centro de massa (CM). O modelo cinemático de um robô diferencial com translação bi-dimensional e rotação no plano é descrito pela expressão (4.1):

$$\begin{cases} \dot{x}(t) = v(t) \cdot cos\left(\theta(t)\right) \\ \dot{y}(t) = v(t) \cdot sen\left(\theta(t)\right) \\ \dot{\theta}(t) = v(t) \cdot \gamma(t) \end{cases}, \tag{4.1}$$

onde:

- $v(t)$: velocidade tangencial do robô no ponto central entre as rodas;
- $\theta(t)$: posição angular do robô em relação ao sistema de coordenadas globais;
- $\gamma(t)$: curvatura do robô;
- $r(t)$: raio de curvatura do robô;
- $(x(t), y(t))$: posição do robô em coordenadas cartesianas globais.

A partir da Figura 4.1, pode-se obter o modelo linearizado em coordenadas locais. Para isso, considera-se que o sistema de coordenadas locais (x_L, y_L) esteja sempre posicionado sobre o robô, de modo que o eixo x_L esteja sempre no sentido do seu comprimento, como mostra a Figura 4.1. A partir desta figura, tem-se:

$$\begin{cases} dx_L = \dfrac{1}{\gamma} \cdot sen(d\theta) \\ dy_L = \dfrac{1}{\gamma} \cdot (1 - cos(d\theta)) \\ \gamma = \dfrac{d\theta}{v(t) \cdot dt} \end{cases},$$

de modo que, substituindo a equação de γ nas equações de x_L e y_L, e depois a manipulando algebricamente, chega-se ao modelo cinemático em coordenadas locais que é dado por:

$$\begin{cases} \dot{x}_L(t) = \dfrac{v}{d\theta} \cdot sen(d\theta(t)) \\ \dot{y}_L(t) = \dfrac{v}{d\theta} \cdot (1 - cos(d\theta(t)) \\ \dot{\theta}(t) = v(t) \cdot \gamma(t) \end{cases}, \tag{4.2}$$

onde a orientação do veículo $\theta(t)$ está no sistema de coordenadas globais.

O modelo cinemático linearizado em sistema de coordenadas locais é obtido a partir do modelo não-linear em coordenadas locais, dado pela expressão (4.2). Para isso, é assumido que a cada amostra, os incrementos do ângulo de orientação $\theta(t)$ da expressão (4.2) são pequenos (NORMEY-RICO *et al.*, 1998). Assim, expandindo em Séries de Taylor $cos(d\theta(t))$ e $sen(d\theta(t))$, e truncando no segundo e no primeiro termo respectivamente, tem-se $cos(d\theta(t)) \simeq 1 - \frac{d\theta(t)^2}{2!}$ e $sen(d\theta(t)) \simeq d\theta(t)$. Portanto, o modelo cinemático em coordenadas locais linearizado é dado por:

$$\begin{cases} \dot{x}_L(t) = v(t) \\ \dot{y}_L(t) = \dfrac{v(t)}{2} \cdot d\theta(t) \\ \dot{\theta}(t) = v(t) \cdot \gamma(t) \end{cases} \tag{4.3}$$

Aplicando o método de discretização de Euler ($\dot{x}(k) = \frac{x(k+1)-x(k)}{T}$), tem-se

$$\begin{cases} x_L(k+1) = x_L(k) + v(k) \cdot T \\ y_L(k+1) = y_L(k) + \dfrac{v(k) \cdot T}{2} \cdot \Delta\theta(k) \\ \theta(k+1) = \theta(k) + v(k) \cdot T \cdot \gamma(k) \end{cases}$$

onde substituindo $\Delta\theta(k)$ por $\theta(k+1) - \theta(k)$, chega-se ao modelo linear discreto:

$$\begin{cases} x_L(k+1) = x_L(k) + v(k) \cdot T \\ y_L(k+1) = y_L(k) + \dfrac{(v(k) \cdot T)^2}{2} \cdot \gamma(k) \\ \theta(k+1) = \theta(k) + v(k) \cdot T \cdot \gamma(k) \end{cases} \tag{4.4}$$

4.3 Modelagem de um Robô Direcional

Robôs móveis com sistema de direção baseado em Estrutura de Ackerman são aqui denominados robôs direcionais, caracterizados por duas rodas traseiras alinhadas com o veículo e duas rodas dianteiras que podem ser esterçadas, sendo portanto, guiáveis

como mostra a Figura 4.2. A modelagem do veículo direcional parte das mesmas considerações de rigidez do corpo e não deformabilidade das rodas. De modo a simplificar a modelagem, os pares de rodas dianteiras e traseiras são representados por apenas uma roda na frente e outra atrás na metade de cada eixo mecânico (MURRAY e SASTRY, 1993). Assim, assume-se que não ocorre escorregamento e, portanto, a velocidade do veículo no centro das duas rodas de cada eixo mecânico é sempre tangente à orientação do mesmo (BARRAQUAND e LATOMBE, 1989). Além disso, sendo que as rodas dianteiras e traseiras possuem a mesma dimensão, pode-se considerar que o módulo da velocidade em ambas é igual. Com isso, as restrições do sistema permitem que as rodas rolem e girem, mas não deslizem.

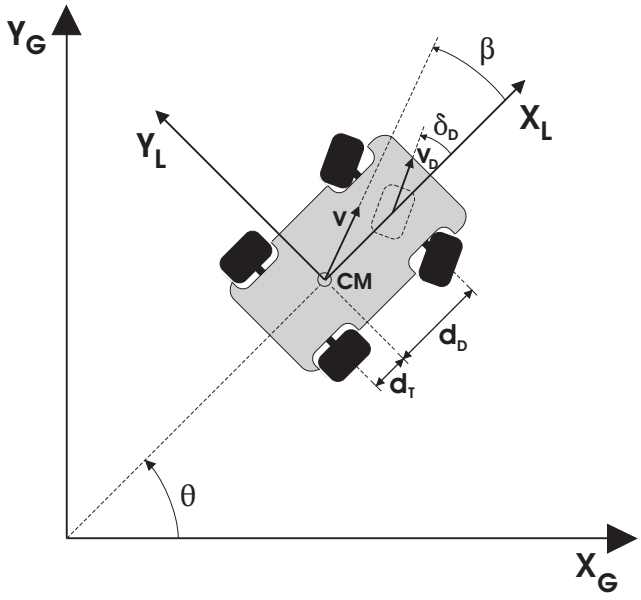

Figura 4.2: Configuração de um robô direcional.

A partir da Figura 4.2, o ponto de guiamento é definido no centro das rodas dianteiras e o centro de rotação está localizado no centro de massa (CM). O modelo cinemático de um robô direcional com translação bi-dimensional e rotação no plano é descrito pela expressão (4.5) (BARRAQUAND e LATOMBE, 1989; NELSON, 1989; MURRAY e SASTRY, 1993; NECSULESCU *et al.*, 1996; SCHAMMASS *et al.*, 1998):

$$\begin{cases} \dot{x}(t) = v_D(t) \cdot cos\left(\theta(t) + \delta_D(t)\right) \\ \dot{y}(t) = v_D(t) \cdot sen\left(\theta(t) + \delta_D(t)\right) \\ \dot{\theta}(t) = \dfrac{v_D(t)}{d_D} \cdot sen\delta_D(t) \end{cases}, \tag{4.5}$$

onde:

- $v_D(t)$: velocidade tangencial do robô no ponto central entre as rodas dianteiras;
- d_D: distância entre o eixo dianteiro e o centro de massa do robô;
- $\theta(t)$: posição angular do robô em relação ao sistema de coordenadas globais;
- $\delta_D(t)$: ângulo da roda dianteira em relação ao eixo x_L fixo ao robô;
- $(x(t), y(t))$: coordenadas cartesianas globais do robô.

A obtenção do modelo cinemático em coordenadas locais do robô direcional parte da mesma consideração utilizada na modelagem cinemática de robôs diferenciais, ou seja, é considerado que o sistema de coordenadas (x_L, y_L) é fixo e localizado no centro de massa do robô e que o eixo x_L é paralelo ao comprimento do robô. Assim, o modelo cinemático em coordenadas locais do robô representado pela Figura 4.2 é dado por:

$$\begin{cases} \dot{x}_L(t) = v_D(t) \cdot cos(\delta_D(t)) \\ \dot{y}_L(t) = v_D(t) \cdot sen(\delta_D(t)) \\ \dot{\theta}(t) = \dfrac{v_D(t)}{d_D} \cdot sen(\delta_D(t)) \end{cases}, \tag{4.6}$$

onde a orientação do veículo $\theta(t)$ está no sistema de coordenadas globais.

Assim como na Seção 4.2, o modelo cinemático linearizado é obtido a partir do modelo em coordenadas locais, dado pela expressão (4.6) no qual os incrementos dos ângulos de orientação do veículo, $\theta(t)$, e de direção da roda dianteira, $\delta_D(t)$, são pequenos a cada amostra. Assim, expandindo, através de Séries de Taylor, $cos(\delta_D(t))$ e $sen(\delta_D(t))$, e truncando as expansões no primeiro termo, tem-se $cos(\delta_D(t)) \simeq 1$ e $sen(\delta_D(t)) \simeq \delta_D(t)$. Portanto, o modelo cinemático em coordenadas locais linearizado é dado por:

$$\begin{cases} \dot{x}_L(t) = v_D(t) \\ \dot{y}_L(t) = v_D(t) \cdot \delta_D(t) \\ \dot{\theta}(t) = \dfrac{v_D(t)}{d_D} \cdot \delta_D(t) \end{cases} \tag{4.7}$$

Utilizando, novamente, o método de Euler para obter o modelo discretizado para o sistema (4.7), tem-se:

$$\begin{cases} x_L(k+1) = x_L(k) + v_D(k) \cdot T \\ y_L(k+1) = y_L(k) + v_D(k) \cdot T \cdot \delta_D(k) \\ \theta(k+1) = \theta(k) + \dfrac{v_D(k) \cdot T}{d_D} \cdot \delta_D(k) \end{cases} \tag{4.8}$$

Assim como o modelo linearizado para robôs diferenciais, o modelo linearizado da expressão (4.8) é válido somente para pequenos valores de incrementos de $\theta(t)$ e $\delta_D(t)$, o que remete novamente à necessidade da geração de rotas de aproximação.

4.3.1 Modelo Dinâmico

Devido ao fato de que robôs e veículos do tipo automóvel freqüentemente se apresentam com dimensões e massa não desprezíveis, a influência das forças que atuam sobre a estrutura destes deve ser considerada em aplicações em que se deseja bom desempenho em velocidades elevadas. *Velocidades elevadas* são aqui consideradas velocidades próximas aos limites de estabilidade dinâmica particulares de cada veículo ou robô. A modelagem da dinâmica do modelo do tipo automóvel será apresentada de forma simplificada, com base no trabalho realizado por LEDUR (2003).

Esta modelagem é desenvolvida a partir das seguintes considerações:

- É considerado que o centro de massa (CM) do robô se encontra ao nível do solo, eliminando a influência de movimentos de rotação em torno dos eixos x e y apresentados na Figura 4.3. Com isso, analisam-se apenas os movimentos de rotação em torno do eixo z e de deslocamento no plano xy.

- Assim como na modelagem dos aspectos cinemáticos, supõe-se que as rodas em cada um dos eixos mecânicos do robô são agrupadas em apenas uma, como mostrado na Figura 4.4. Assim, desprezam-se todos os tipos de movimentos oscilatórios ou rotatórios em torno do eixo x. O efeito da suspensão do robô também é desconsiderado na modelagem dinâmica, ou seja, sem o efeito massa-mola-amortecedor o robô é considerado como um retângulo se movendo sobre o solo.

Ao desprezar o efeito da suspensão do veículo, elimina-se também a influência da variação de carga sobre os pneus, e assim a massa do robô é distribuída uniformemente ao longo do chassi.

- No diagrama de forças do robô, considera-se que o mesmo possui tração nos eixos traseiro e dianteiro para facilitar a adaptação de diversas formas de tração. Caso o robô não possua algum destes componentes, basta anulá-lo.

Figura 4.3: Rotação e deslocamento nos eixos xyz.

A partir do diagrama de forças representado na Figura 4.5, são obtidas as equações da dinâmica do robô. Nestas, são consideradas todos as forças de interesse em relação ao sistema de coordenadas locais, bem como os ângulos e distâncias das mesmas com relação ao centro de massa.

Através do diagrama, pode-se observar que as forças representadas atuam em quatro diferentes pontos, sendo eles o centro de massa (CM), o centro de convergência aerodinâmica (CA) e os pontos de contato dos pneus dianteiro e traseiro com o solo (LEDUR, 2003).

O vetor velocidade v, que rotaciona em relação ao eixo x_L de um ângulo β, e o vetor força centrífuga F_{CF}, que é perpendicular ao vetor velocidade e que tende a impulsionar o robô para fora do eixo de rotação em torno do eixo z_L, estão localizados no CM. O

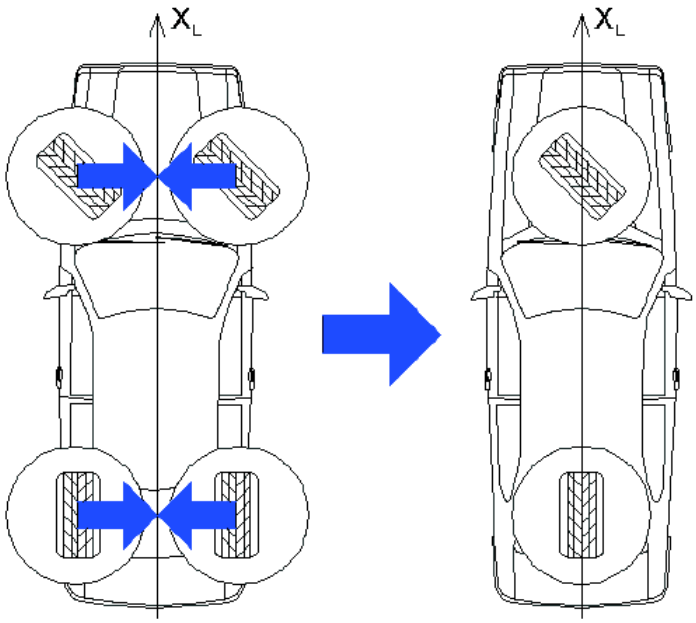

Figura 4.4: Agrupamentos das rodas.

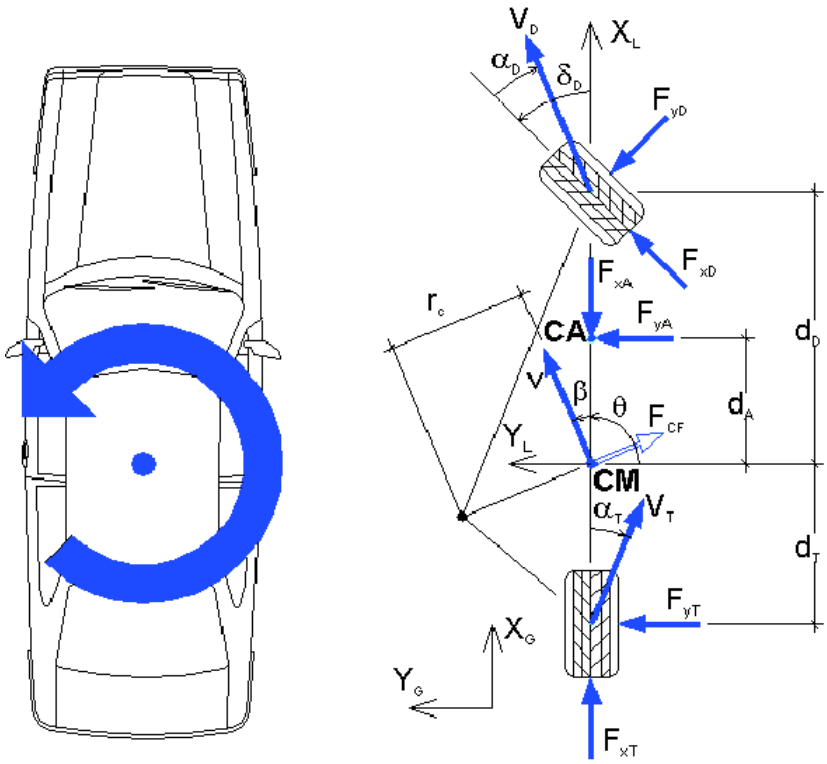

Figura 4.5: Diagrama de forças de um robô direcional.

ângulo θ descreve o deslocamento angular, como já mencionado anteriormente, do eixo x_L fixo ao robô em relação ao eixo x do sistema de coordenadas globais. No CA está

sendo aplicada a força aerodinâmica (F_A), que é decomposta sobre os eixos x_L e y_L em F_{xA} e F_{yA}, respectivamente. F_{xA} e F_{yA} descrevem, de forma simplificada, todo o efeito aerodinâmico que atua sobre o robô.

Na roda traseira, estão localizadas a força de tração traseira F_{xT}, que é uma das forças responsáveis pelo deslocamento do robô, e a força de aderência lateral do pneu traseiro F_{yT}, que atua no sentido contrário ao deslocamento lateral do robô. Neste ponto de atuação de forças, também se encontra o vetor de deslocamento lateral V_T, que está inclinado em relação ao eixo x_L pelo ângulo α_T. Este vetor representa a tendência do robô permanecer em uma trajetória retilínea (RAFFO, 2005).

As forças atuantes sobre a roda dianteira são definidas da mesma maneira que as atuantes na roda traseira, porém a incidência das primeiras é determinada pelo ângulo δ_D que descreve o ângulo de esterçamento da roda dianteira em relação ao eixo x_L.

No diagrama de forças, também estão descritas as distâncias entre o CM e os demais pontos de atuação de forças; com o auxílio destas distâncias se define a equação de momento torsor em relação ao centro de massa do robô.

A partir da definição das forças relevantes que atuam sobre os eixos x_L e y_L e dos torques que atuam sobre o eixo z_L, em LEDUR (2003), são obtidas, com base na Segunda Lei de Newton nas versões para corpos em translação e rotação, as equações dinâmicas de β e $\dot{\theta}$. Em RAFFO (2005), estas equações são linearizadas com base nas considerações propostas por KELBER *et al.* (2004), chegando-se às expressões (4.9) e (4.10).

$$\dot{\beta} = \dot{\theta} \cdot \left(\frac{c_T \cdot d_T - c_D \cdot d_D}{m \cdot v^2} - 1 \right) - \beta \cdot \left(\frac{c_T + c_D}{m \cdot v} \right) + \frac{F_{xD} \cdot \delta_D}{m \cdot v} + \frac{c_D \cdot \delta_D}{m \cdot v} \tag{4.9}$$

$$\begin{aligned} \ddot{\theta} &= \frac{\beta}{J_z} \cdot (c_T \cdot d_T - c_D \cdot d_D) - \frac{\dot{\theta}}{J_z \cdot v} \cdot \left(c_T \cdot d_T^2 + c_D \cdot d_D^2 \right) + \frac{c_D \cdot d_D \cdot \delta_D}{J_z} \\ &+ \frac{F_{xD} \cdot d_D \cdot \delta_D}{J_z} \end{aligned} \tag{4.10}$$

Já a dinâmica da velocidade v, conforme é mostrado em GOMES (2003) e em KEL-

BER *et al.* (2004), apresenta comportamento diferenciado nas situações de aceleração e frenagem, devido à natureza distinta dos sistemas eletromecânicos envolvidos em cada opreração. A dinâmica linearizada da velocidade v para cada uma das situações foi identificada, experimentalmente, em GOMES (2003), para um veículo direcional do tipo Mini-Baja com tração traseira, por meio da resposta a degraus de aceleração e frenagem. Para isso, considera-se que os ângulos de orientação e do vetor velocidade se mantêm sobre a referência e o motor a combustão opera na região de regime. A função de transferência simplificada obtida para a situação de aceleração é dada pela expressão (4.11)

$$G_{vel}(s)_{acel} = \frac{v(s)}{\varphi(s)} = \frac{V_{Vacel}}{T_{Mgas} \cdot T_V \cdot s^2 + (T_{Mgas} + T_V) \cdot s + 1} \tag{4.11}$$

onde φ é o ângulo de abertura do carburador, T_{Mgas} é a constante de tempo do modelo simplificado do motor, T_V é a constante de tempo do veículo, que relaciona a massa deste com os atritos impostos a ele, e V_{Vacel} é o ganho que relaciona a força necessária para deslocar o veículo com os atritos aos quais este é submetido.

A função de transferência simplificada obtida para a frenagem é dada pela expressão (4.12)

$$G_{vel}(s)_{fren} = \frac{v(s)}{\tau_F(s)} = \frac{V_{Vfren}}{T_1 \cdot s^3 + T_2 \cdot s^2 + T_3 \cdot s + 1} \tag{4.12}$$

onde τ_F é o torque imposto pelo motor de controle do sistema de freios, $T_1 = (T_F \cdot T_{Fmec} \cdot T_V)$, $T_2 = (T_F \cdot T_{Fmec} + T_F \cdot T_V + T_{Fmec} \cdot T_V)$, $T_3 = (T_F + T_{Fmec} + T_V)$, T_F é a constante de tempo do sistema eletrônico de controle de torque de frenagem, T_{Fmec} é a constante de tempo da estrutura mecânica do sistema de freios, T_V é a constante de tempo do veículo, que relaciona a massa deste com os atritos impostos a ele, e V_{Vfren} é o ganho que relaciona o torque de frenagem com a força produzida pelo sistema de freios.

Com isso, modelo dinâmico linearizado de robôs e veículos do tipo automóvel é aqui descrito pelas expressões (4.9, 4.10, 4.11, 4.12) de modo que o controle do comporta-

mento dinâmico destes pode, sob certas restrições, ser controlado utilizando técnica de CPBM linear.

4.4 Transformação de Coordenadas

Para poder-se utilizar o controlador baseado no modelo em coordenadas locais, é necessário transformar o caminho de referência também para as coordenadas locais, pois a trajetória de referência é pré-determinada no sistema de coordenadas globais. Assim, a cada período amostral em que se recebe a trajetória de referência futura, esta é transformada para o sistema de coordenadas locais, cuja orientação corresponde à posição atual do robô, como é apresentado na Figura 4.6, onde $\alpha = (\phi - \theta(k))$ e $d = \sqrt{(x_{ref}(k+j) - x(k))^2 + (y_{ref}(k+j) - y(k))^2}$ e j é o índice que define o número de amostras à frente do passo k.

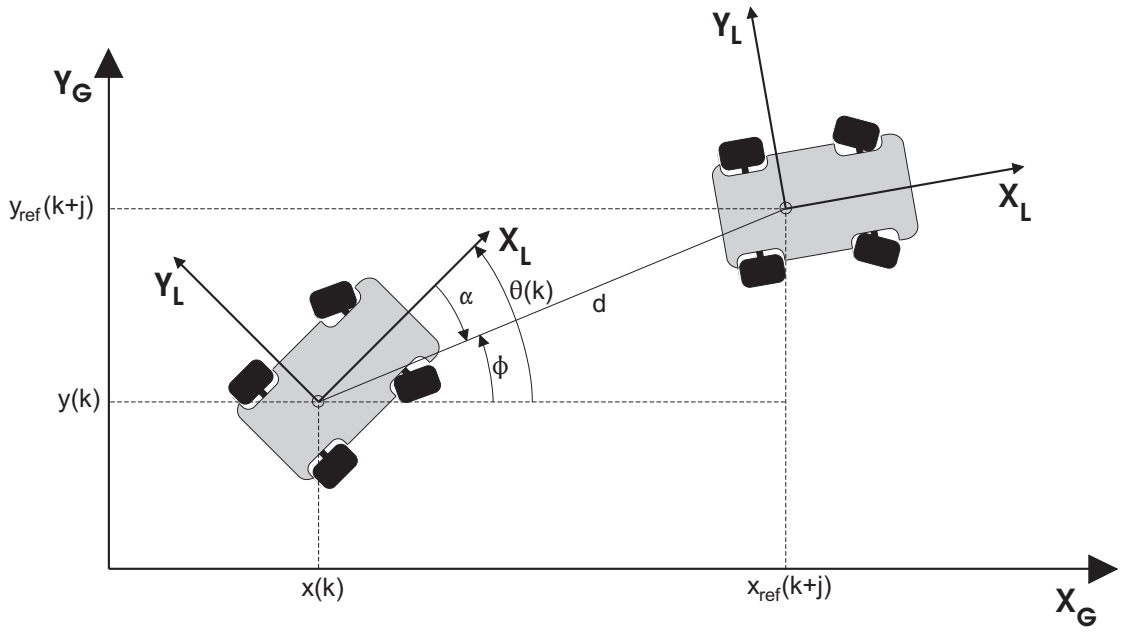

Figura 4.6: Transformação de coordenadas.

Assim, o ângulo ϕ é obtido por:

$$\phi = arctg\left(\frac{y_{ref}(k+j) - y(k)}{x_{ref}(k+j) - x(k)}\right) \quad , \tag{4.13}$$

e as coordenadas locais do ponto $(x_{ref}(k+j), y_{ref}(k+j))$ em relação à posição atual

do robô são:

$$\begin{aligned} x_{Lref}(k+j) &= d \cdot cos\,(\phi - \theta(k)) \\ y_{Lref}(k+j) &= d \cdot sen\,(\phi - \theta(k)) \end{aligned} \tag{4.14}$$

Para realizar a transformação de coordenadas locais para globais, faz-se:

$$\alpha = arctg\left(\frac{y_L(k+j)}{x_L(k+j)}\right) \tag{4.15}$$

$$d = \sqrt{x_L(k+j)^2 + y_L(k+j)^2} \tag{4.16}$$

$$\begin{aligned} x(k+j) &= d \cdot cos\,(\theta(k) - \alpha) + x(k) \\ y(k+j) &= d \cdot sen\,(\theta(k) - \alpha) + y(k) \end{aligned} \tag{4.17}$$

Tais transformações são utilizadas tanto para gerar a trajetória de aproximação, quanto para colocá-la no sistema de trabalho.

4.5 Trajetória de Aproximação

Vários métodos de seguimento de trajetória são baseados em encontrar o erro entre a posição atual do robô e a desejada sobre uma trajetória de referência. Porém, se o veículo não está sobre a referência, é necessário aproximá-lo desta utilizando trajetórias de aproximação (NORMEY-RICO *et al.*, 1999). Estratégias de aproximação, em geral, são compostas pelos seguintes passos (AMIDI, 1990):

- escolher um ponto a uma distância qualquer à frente do veículo sobre a trajetória de referência como o ponto de destino;
- usar a informação deste ponto para definir o caminho a ser seguido.

Em AMIDI (1990) foram testados alguns algoritmos que geram trajetória de aproximação. Dentre eles estão o *CTA* (*Control Theory Approach*), o *QPF*(*Quintic Polynomial Fit*) e o *pure pursuit*. O *CTA* apresenta raio de curvatura muito pequeno, que é

aproximadamente o menor raio de giro possível, devido a questões físicas, que o veículo pode realizar. O método, apesar de ser bastante simples, apresenta boa performance. Porém, quando utilizado para superar pequenos erros de orientação, resulta em pequenas oscilações no ângulo de direção da roda dianteira, as quais não são desejadas em aplicações da alto desempenho. O *QPF* tem como finalidade evitar movimentos descontínuos do ângulo de direção da roda dianteira, mas a complexidade e o difícil entendimento deste algoritmo se caracterizam como obstáculos relevantes. O *pure pursuit* é um algoritmo de fácil implementação, pois considera somente a localização do veículo e do ponto de destino, além de gerar trajetórias de aproximação bastante estáveis e precisas (AMIDI, 1990).

Em WIT *et al.* (2004), foi apresentada e testada a técnica *vector pursuit*, baseada na teoria dos helicóides desenvolvida na trabalho pioneiro de Sir Robert Stawell Ball em 1900 e apresentada em BALL (1900). A técnica inova no sentido de que considera geometricamente a orientação de referência na posição à frente do veículo sobre a trajetória. Como resultado, o *vector pursuit* proporciona maior robustez frente a variações na distância para a escolha do ponto à frente, se comparada a técnicas como o *pure pursuit*.

Com base nos resultados apresentados em diversos trabalhos citados anteriormente e no fato de que o trabalho desta dissertação já estava em andamento quando foi apresentada a técnica do *vector pursuit*, o algoritmo *pure pursuit* foi escolhido para ser usado no controlador projetado, como comentado no Capítulo 2. O *pure pursuit* foi originalmente desenvolvido como um método para calcular o arco necessário para colocar o veículo sobre a trajetória de referência (COUTLER, 1992). Esta primeira aplicação do método foi realizada com o *Terragator*, um robô com seis rodas desenvolvido nos anos oitenta. *pure pursuit* é um algoritmo de seguimento de trajetória que trabalha calculando a curvatura na qual o veículo se moverá a partir da sua posição atual até um ponto de destino. Este se encontra sobre a trajetória de referência e está distante do veículo por uma distância denominada *look-ahead* em AMIDI (1990), a qual pode ser fixa ou alterada *on-line* de forma periódica. O *look-ahead* é análogo à distância visualizada por um motorista à frente do carro, e por isso, ao seguir o caminho, arcos para diferentes pontos de destino vão sendo gerados, à medida que o veículo se move,

como mostra a Figura 4.7.

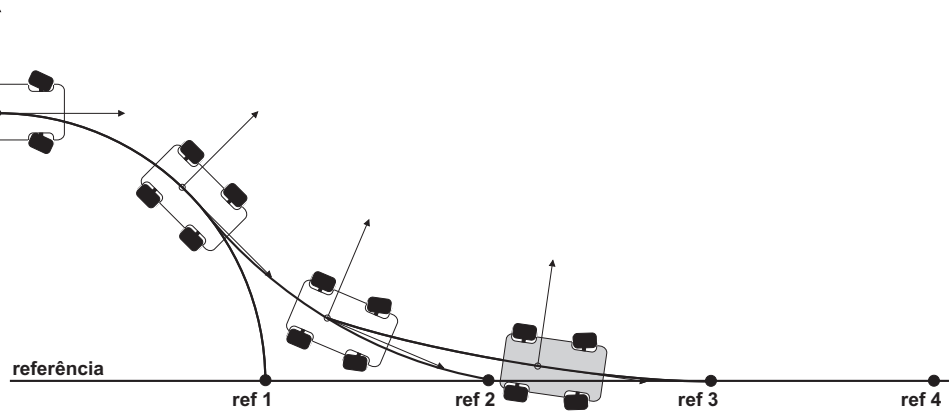

Figura 4.7: Seqüência de arcos do *pure pursuit*.

A escolha do *look-ahead* deve considerar dois problemas: alcançar uma trajetória e, após isso, manter-se sobre ela. No primeiro problema, se o *look-ahead* é muito longo, o veículo tende a convergir para o caminho gradualmente e com menos oscilação, ou seja, de maneira mais estável. Assim, pode-se relacionar a resposta do *pure pursuit* com a resposta ao degrau de um sistema dinâmico de segunda ordem e o valor do *look-ahead* como um fator de amortecimento. No segundo problema, um *look-ahead* muito grande diminui a capacidade de curvatura do robô e, em conseqüência disso, faz com que um caminho menos curvilíneo seja seguido. O algoritmo calcula uma curvatura para que o robô possa percorrer um arco; porém, se o caminho entre o veículo e o ponto de destino é muito curvilíneo, então existe um arco não singular que une os dois pontos que induzirá a um erro de direcionamento do veículo (COUTLER, 1992). Além desses problemas, o *look-ahead* deve ser escolhido considerando diferentes velocidades. Uma maneira para encontrar esta distância pode ser experimental, com a finalidade de encontrar o relacionamento do *look-ahead* com a velocidade (AMIDI, 1990). Neste trabalho, foi utilizado um método adaptativo para determinar esta distância, ou seja, em um período pré-definido é calculada a distância entre a posição atual do veículo e o ponto de destino sobre a trajetória de referência e, dependendo desta, o *look-ahead* é recalculado.

O raio de curvatura é calculado considerando as coordenadas do veículo e da referência no sistema fixo ao robô, como é apresentado na Figura 4.8. O par $(x_{Lref}(k+j), y_{Lref}(k+j))$ são as coordenadas locais do ponto $(x_{ref}(k+j), y_{ref}(k+j))$

da trajetória de referência em relação à posição atual do veículo $(x(k), y(k))$, L é o *look-ahead* e r é o raio do arco a ser construído.

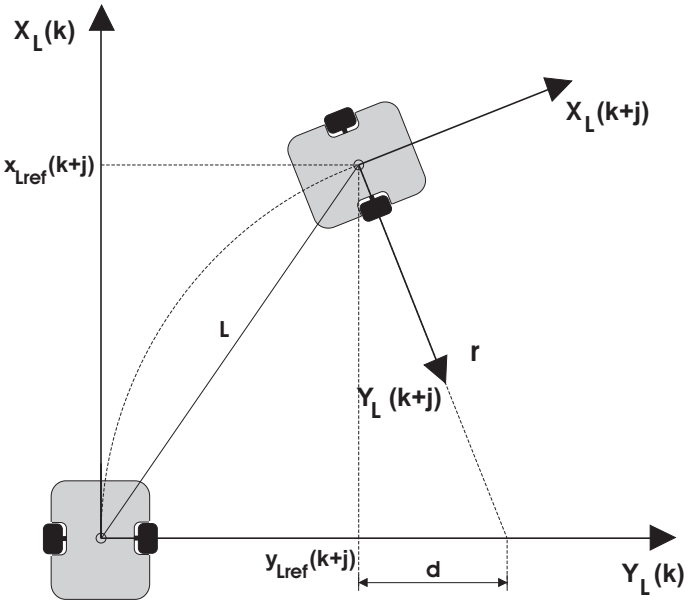

Figura 4.8: Geometria do Algoritmo.

Assim, tem-se:

$$x_{Lref}(k+j)^2 + y_{Lref}(k+j)^2 = L^2$$

$$d = r - y_{Lref}(k+j)$$

$$(r - y_{Lref}(k+j))^2 + x_{Lref}(k+j)^2 = r^2$$

$$r = \frac{L^2}{2y_{Lref}(k+j)} \tag{4.18}$$

Com isso, para cada ponto da trajetória de referência distante pelo *look-ahead* do veículo, um arco com raio r é calculado. O veículo então se aproximará suavemente do caminho de destino, sem que ocorram incrementos elevados no ângulo de orientação do robô.

O método apresenta algumas limitações relacionadas aos efeitos da dinâmica do veículo, já que a capacidade deste ou de seus atuadores não é modelada e assume-se

que a resposta é perfeita para as curvaturas exigidas. Estas limitações fazem com que o veículo saia de traseira e derrape quando ocorre uma mudança brusca na curvatura e este está em alta velocidade (RAFFO, 2005). Além disso, o veículo não se aproximará da trajetória tão rapidamente quanto desejado por causa do atraso na resposta da direção (COUTLER, 1992).

4.6 GPC Aplicado a Seguimento de Trajetória

Como foi descrito na Sessão 4.1, em RAFFO (2005) e NORMEY-RICO *et al.* (1999), mostrou-se que o algoritmo GPC pode ser aplicado ao problema de seguimento de trajetória de robôs móveis nos níveis de dinâmica e condução. O controle de orientação e posição de robôs móveis considerando somente o modelo cinemático é válido quando estes trabalham em baixas velocidades, com pequenas acelerações e os efeitos da massa do veículo e da carga que este transporta possam ser desconsideradas. A Figura 4.9 apresenta a estrutura do sistema de controle de seguimento de trajetória apenas para o modelo cinemático.

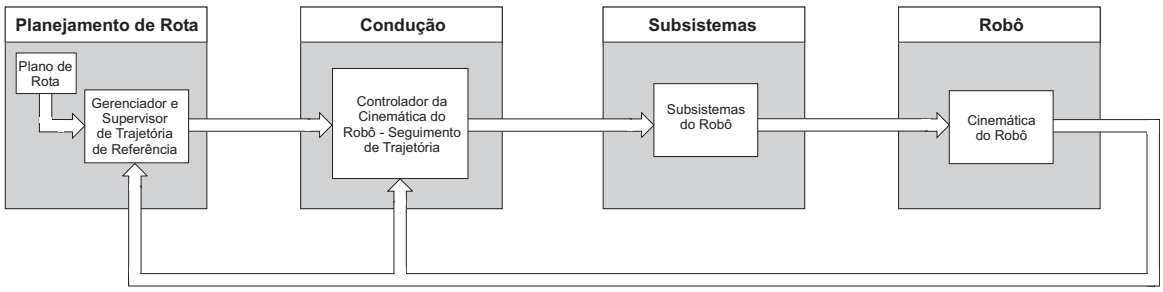

Figura 4.9: Arquitetura de controle baseado em modelo cinemático

No entanto, como comentado no Capítulo 2 e na Sessão 4.3, quando veículos autônomos são designados a obter performance em tarefas de alto desempenho e percorrer caminhos em altas velocidades, o modelo dinâmico deles se torna muito importante (BOYDEN e VELINSKY, 1994) e deve ser considerado. A Figura 4.10 apresenta a estrutura de controle considerando também o modelo dinâmico.

Nesta seção é apresentado o *Generalized Predictive Control* (GPC), bem como a sua aplicação aos modelos dinâmicos e cinemáticos de robôs móveis, utilizando a estrutura das figuras 4.9 e 4.10 para tratar o problema de seguimento de trajetória.

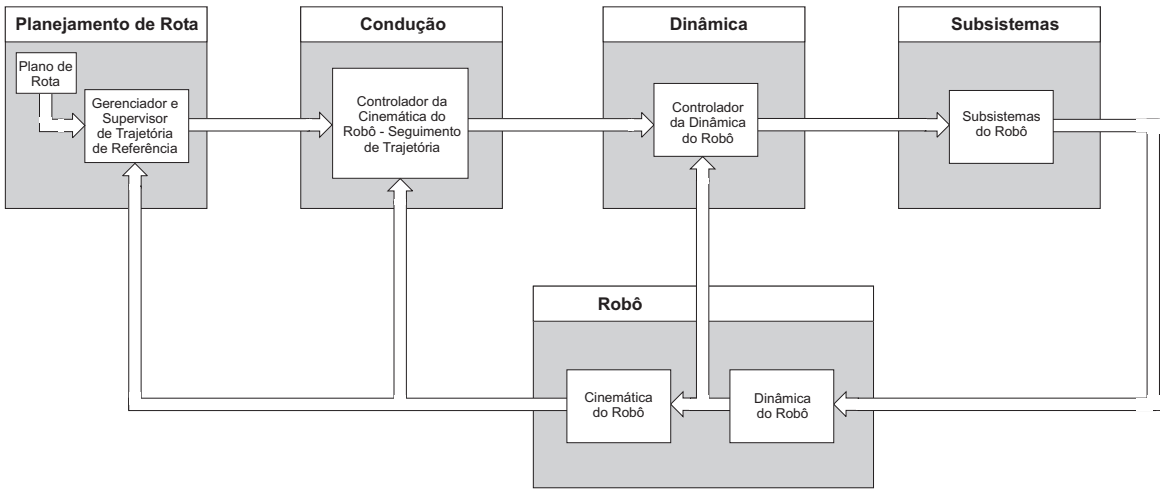

Figura 4.10: Arquitetura de controle em cascata considerando modelo dinâmico

4.6.1 O Controle Preditivo Generalizado - GPC

O controle preditivo generalizado (CLARKE *et al.*, 1987), ou GPC, é uma técnica de CPBM linear que utiliza modelo baseado em função de transferência, sendo capaz de tratar processos multivariáveis. O modelo multivariável discreto utilizado pelo GPC é dado por uma função de transferência $n \times m$ $\mathbf{P}(z^{-1})$:

$$\mathbf{y}(k) = \mathbf{P}(z^{-1}) \cdot \mathbf{u}(k) \tag{4.19}$$

onde $\mathbf{u}(k)$ é o vetor de entradas $m \times 1$ e $\mathbf{y}(k)$ é o vetor de saídas $n \times 1$. Cada elemento $P_{pq}(z^{-1})$ de $\mathbf{P}(z^{-1})$ é uma função de transferência SISO:

$$\begin{aligned} P_{pq}(z^{-1}) &= \frac{z^{-1}B'_{pq}(z^{-1})}{A_{pq}(z^{-1})} \\ &= \frac{z^{-1}B_{pq}(z^{-1})}{A_{p}(z^{-1})} = g_{pq}(z^{-1}) \end{aligned} \tag{4.20}$$

onde $A_{pq}(z^{-1})$ e $B'_{pq}(z^{-1})$ são, respectivamente, o denominador e o numerador da função de transferência entre a entrada q e a saída p, e $A_p(z^{-1})$ é o mínimo múltiplo comum dos polinômios $A_{pq}(z^{-1})$ para $q = 1 \cdots m$ e $p = 1 \cdots n$. Considerando a matriz polinomial diagonal $\mathbf{A}(z^{-1})$ com os elementos $A_p(z^{-1})$ e as matrizes polinomiais $\mathbf{B}(z^{-1})$ e $\mathbf{G}(z^{-1})$

com os elementos $B_{pq}(z^{-1})$ e $g_{pq}(z^{-1})$, respectivamente, o modelo do processo é descrito por:

$$\mathbf{P}(z^{-1}) = \mathbf{G}(z^{-1}) \tag{4.21}$$

Como o algoritmo GPC usa o modelo *Controlled Auto-Regressive and Integrated Moving Average* (CARIMA), a partir do processo multivariável (4.19) com n-saídas e m-entradas, ele é escrito da seguinte forma:

$$\mathbf{A}\left(z^{-1}\right) \cdot \mathbf{y}(k) = \mathbf{B}\left(z^{-1}\right) \cdot \mathbf{u}(k-1) + \frac{\mathbf{C}\left(z^{-1}\right)}{\Delta} \cdot \mathbf{e}(k) \tag{4.22}$$

com

$$\begin{aligned} A_p\left(z^{-1}\right) &= 1 + a_1^p z^{-1} + a_2^p z^{-2} + \cdots + a_{na_p}^p z^{-na_p} \\ B_{pq}\left(z^{-1}\right) &= b_0^{pq} + b_1^{pq} z^{-1} + b_2^{pq} z^{-2} + \cdots + b_{nb_{pq}}^{pq} z^{-nb_{pq}} \end{aligned} \tag{4.23}$$

Por questão de simplicidade e pelo fato de que, na prática, os polinômios coloridos são muito difíceis de serem estimados com boa precisão, escolhe-se $\mathbf{C}(z^{-1}) = I_{n\times n}$. O operador Δ é definido como $\Delta = 1 - z^{-1}$ e $\mathbf{e}(k)$ é o vetor de ruídos $n \times 1$, que se considera ser um ruído branco com média zero. Assim, o modelo (4.22) é reescrito (CAMACHO e BORDONS, 1998; NORMEY-RICO e CAMACHO, 2000):

$$\mathbf{A}\left(z^{-1}\right) \cdot \mathbf{y}(k) = \mathbf{B}\left(z^{-1}\right) \cdot \mathbf{u}(k-1) + \frac{1}{\Delta} \cdot \mathbf{e}(k) \tag{4.24}$$

Como a matriz $\mathbf{A}(z^{-1})$ é diagonal, as equações Diofantinas necessárias à obtenção da predição ótima de cada saída também podem ser resolvidas independentemente para cada uma delas. Assim, o modelo CARIMA (4.24) expresso para uma saída é dado por:

$$A_p\left(z^{-1}\right) \cdot y_p(k) = \mathbf{B}_p\left(z^{-1}\right) \cdot \mathbf{u}(k-1) + \frac{e(k)}{\Delta} \tag{4.25}$$

onde $\mathbf{B}_p = [B_{p1}\ B_{p2}\ \cdots\ B_{pm}]$. A saída predita $\hat{y}_p(k+j|k)$, para amostras j à frente de k, é obtida da solução da seguinte equação Diofantina:

$$1 = E_{pj}(z^{-1})A_p(z^{-1})\Delta + z^{-j}F_{pj}(z^{-1}) \tag{4.26}$$

onde $E_{pj}(z^{-1})$ e $F_{pj}(z^{-1})$ são polinômios de ordem $j-1$ e na_p, respectivamente.

Se (4.25) é multiplicado por $\Delta E_{pj}(z^{-1})z^j$ tem-se:

$$E_{pj}(z^{-1})\tilde{A}_p(z^{-1})y_p(k+j) = E_{pj}(z^{-1})\mathbf{B}(z^{-1})\Delta\mathbf{u}(k+j-1) + E_{pj}(z^{-1})e(k+j) \tag{4.27}$$

com $\tilde{A}_p(z^{-1}) = \Delta A_p(z^{-1})$.

Da expressão (4.27), é tido que no segundo membro da igualdade apenas os termos do ruído futuro não são conhecidos. Também devido ao fato de que o grau de $E_j(z^{-1})$ é $j-1$, tem-se que os termos dependentes do ruído estão no futuro. Como no sentido estocástico a melhor predição de $y_p(k+j|k)$ é aquela que considera como nulos os valores do ruído futuro (CAMACHO e BORDONS (1998); NORMEY-RICO e CAMACHO (2000)), fazendo-se $e(k+j) = 0$ para $j > 0$ e realizando-se algumas manipulações matemáticas, obtém-se a predição ótima da saída $\hat{y}_p(k+j|k)$:

$$\hat{y}_p(k+j|k) = E_{pj}(z^{-1})\mathbf{B}_p(z^{-1})\Delta\mathbf{u}(k+j-1|k) + F_{pj}(z^{-1})y_p(k)\ . \tag{4.28}$$

A expressão (4.28) pode ser escrita na forma matricial:

$$\hat{\mathbf{y}}_p = \mathbf{G}_p\Delta\mathbf{u} + \mathbf{f}_p\ , \tag{4.29}$$

onde

$$\hat{\mathbf{y}}_p = \begin{bmatrix} \hat{y}_p(k+1|k) \\ \hat{y}_p(k+2|k) \\ \vdots \\ \hat{y}_p(k+N_{2p}|k) \end{bmatrix}, \quad \mathbf{G}_p = \begin{bmatrix} G_{p1} & G_{p2} & \cdots & G_{pm} \end{bmatrix} \quad ;$$

$$\Delta\mathbf{u} = \begin{bmatrix} \Delta\mathbf{u}_1 \\ \Delta\mathbf{u}_2 \\ \vdots \\ \Delta\mathbf{u}_m \end{bmatrix}, \quad \mathbf{f}_p = \begin{bmatrix} f_p(1) \\ f_p(2) \\ \vdots \\ f_p(N_{2p}) \end{bmatrix},$$

com

$$\Delta\mathbf{u}_q = \begin{bmatrix} \Delta u_q(k) \\ \Delta u_q(k+1) \\ \vdots \\ \Delta u_q(k+N_{uq}-1) \end{bmatrix}, \tag{4.30}$$

$$f_p(j) = z(1-\tilde{A}_p(z^{-1}))f_p(j-1) + \mathbf{B}_p(z^{-1})\Delta\mathbf{u}(k+j)\ . \tag{4.31}$$

e, conforme CAMACHO e BORDONS (1998), devido às propriedades recursivas do polinômio E_{pj}, tem-se:

$$\mathbf{G}_{pq} = \begin{bmatrix} g_{pq0} & 0 & \cdots & 0 \\ g_{pq1} & g_{pq0} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_{pqN_{2p}-1} & g_{pqN_{2p}-2} & \cdots & g_{pqN_{2p}-N_{uq}-1} \end{bmatrix}, \tag{4.32}$$

Os valores de $\Delta\mathbf{u}(k+j)$ quando $j \geq 0$ são iguais a zero e $\mathbf{f}_p(0) = y_p(k)$.

Assim, a predição ótima para um processo MIMO é dada por:

$$\begin{bmatrix} \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \\ \hat{\mathbf{y}}_n \end{bmatrix} = \begin{bmatrix} \mathbf{G}_{11} & \mathbf{G}_{12} & \cdots & \mathbf{G}_{1m} \\ \mathbf{G}_{21} & \mathbf{G}_{21} & \cdots & \mathbf{G}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{G}_{n1} & \mathbf{G}_{n2} & \cdots & \mathbf{G}_{nm} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u}_1 \\ \Delta\mathbf{u}_2 \\ \vdots \\ \Delta\mathbf{u}_m \end{bmatrix} + \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_m \end{bmatrix} \tag{4.33}$$

O algoritmo GPC MIMO consiste na aplicação de uma seqüência de controle que minimiza a seguinte função custo multi-estágio (NORMEY-RICO e CAMACHO, 2000):

$$\mathbf{J}(N_2,\ N_u) = \sum_{j=1}^{N_2} \left\|\hat{\mathbf{y}}(k+j|k) - \mathbf{y}_{ref}(k+j)\right\|_{\mathbf{Q}}^2 + \sum_{j=1}^{N_u} \left\|\Delta\mathbf{u}(k+j-1)\right\|_{\mathbf{R}}^2 \tag{4.34}$$

Portanto, substituindo as predições obtidas (4.33) na expressão (4.34), esta pode ser reescrita como:

$$\mathbf{J} = \left(\mathbf{G}\Delta\mathbf{u} + \mathbf{f} - \mathbf{y}_{ref}\right)' \mathbf{Q} \left(\mathbf{G}\Delta\mathbf{u} + \mathbf{f} - \mathbf{y}_{ref}\right) + \Delta\mathbf{u}'\mathbf{R}\Delta\mathbf{u} \tag{4.35}$$

O mínimo de J, assumindo que não existem restrições no sinal de controle, pode ser obtido analiticamente fazendo $\partial J/\partial\Delta u = 0$, da qual se obtém a lei de controle linear na forma:

$$\Delta\mathbf{u} = \left(\mathbf{G}' \cdot \mathbf{Q} \cdot \mathbf{G} + \mathbf{R}\right)^{-1} \cdot \mathbf{G}' \cdot \mathbf{Q} \cdot \left(\mathbf{y}_{ref} - \mathbf{f}\right) \tag{4.36}$$

Devido à estratégia de horizonte deslizante, somente $\Delta\mathbf{u}(k)$ é necessário no instante k. Assim, para cada $\Delta\mathbf{u}_j$, apenas a primeira linha de $\left(\mathbf{G}' \cdot \mathbf{Q} \cdot \mathbf{G} + \mathbf{R}\right)^{-1} \cdot \mathbf{G}' \cdot \mathbf{Q}$, denominada o ganho K, precisa ser calculada por período amostral, a qual está separada para cada entrada por $1 + \sum_{p=0}^{j-1} N_u(p)$ linhas. Isto pode ser feito *off-line* para um caso não adaptativo. A lei de controle pode, portanto, ser expressa como:

$$\Delta\mathbf{u}(k) = \mathbf{K} \cdot \left(\mathbf{y}_{ref} - \mathbf{f}\right) \tag{4.37}$$

isto é, uma matriz de ganho linear que multiplica os erros preditos entre a referência futura e a resposta livre predita do processo.

Porém, no caso de controle adaptativo, a matriz $\mathbf{G}$ tem que ser calculada a cada amostra, pois os parâmetros estimados mudam. Os cálculos dos incrementos de controle futuros são resolvidos pela equação linear $(\mathbf{G}' \cdot \mathbf{Q} \cdot \mathbf{G} + \mathbf{R}) \cdot \Delta\mathbf{u} = \mathbf{G}' \cdot \mathbf{Q} \cdot \big(\mathbf{y}_{ref} - \mathbf{f}\big)$. Novamente, somente a primeira linha da expressão acima precisa ser computada para cada $\Delta\mathbf{u}_j$, a qual está separada para cada entrada por $1 + \sum_{p=0}^{j-1} N_u(p)$ linhas. Como, nesse caso, realizar inversão matricial *on-line* demanda alto custo computacional, pode-se usar métodos numéricos de solução de equações lineares para inverter matrizes, como, por exemplo, o algoritmo de *Cholesky*, *LU*, *QR*, entre outros (CAMACHO e BORDONS, 1998).

4.6.2 GPC Aplicado ao Modelo Cinemático

A seguir, será apresentada a aplicação do algoritmo GPC ao controle de cinemática em robôtica móvel. O controle da cinemática de veículos autônomos utilizando técnicas de controle preditivo já foi previamente abordado na literatura (BERLIN e FRANK, 1991; NORMEY-RICO *et al.*, 1998, 1999; KIM *et al.*, 2001; van ESSEN e NIJMEIJER, 2001; GU e HU, 2002; KÜHNE *et al.*, 2004). Esta estratégia se torna vantajosa quando o robô percorre uma trajetória previamente estipulada, de modo que, como destacado em CAMACHO e BORDONS (1998), o controle preditivo é bastante favorável em robótica e em processos em lote quando a trajetória de referência futura é conhecida *a priori*.

Na Figura 4.11, é apresentada a estrutura do GPC aplicado ao modelo cinemático. Esta estrutura de aplicação será detalhada separadamente para modelos cinemáticos diferenciais e direcionais.

Modelo Cinemático Diferencial

Para detalhar a aplicação do GPC ao modelo cinemático diferencial, deve-se inicialmente retomar o modelo linearizado obtido na Seção 4.2, dado pela expressão 4.4. Este modelo é apresentado novamente a seguir:

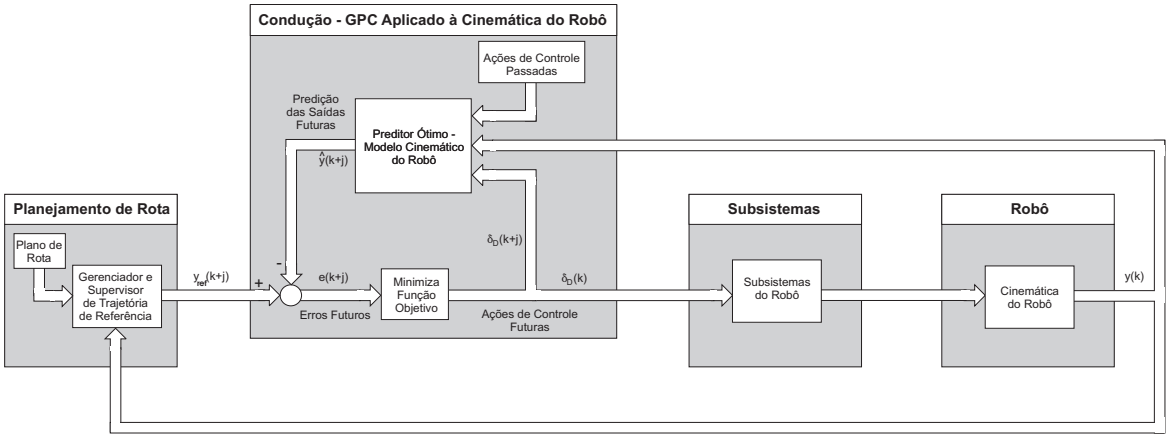

Figura 4.11: GPC aplicado ao modelo cinemático.

$$\begin{cases} x_L(k+1) = x_L(k) + v(k) \cdot T \\ y_L(k+1) = y_L(k) + \dfrac{(v(k) \cdot T)^2}{2} \cdot \gamma(k) \\ \theta(k+1) = \theta(k) + v(k) \cdot T \cdot \gamma(k) \end{cases}$$

Para se utilizar este modelo no algoritmo GPC, deve-se descrevê-lo através de função de transferência, onde se tem a relação entre as variáveis controladas, $\theta(k)$ e $y_L(k)$, e a variável manipulada, $\gamma(k)$. Assim, o modelo cinemático linear em coordenadas locais pode ser escrito da seguinte forma:

$$\left(1 - z^{-1}\right) \cdot \begin{bmatrix} \theta(k) \\ y_L(k) \end{bmatrix} = v(k-1) \cdot T \cdot \begin{bmatrix} 1 \\ (v(k-1) \cdot T)/2 \end{bmatrix} \cdot \gamma(k-1) \tag{4.38}$$

com v_D considerada constante entre duas amostras.

O modelo CARIMA (4.24) para o modelo (4.38) é dado por:

$$\begin{bmatrix} \theta(k) \\ y_L(k) \end{bmatrix} = \frac{v(k-1) \cdot T}{(1 - z^{-1})} \cdot \begin{bmatrix} 1 \\ (v(k-1) \cdot T)/2 \end{bmatrix} \cdot \gamma(k-1) + \frac{1}{(1 - z^{-1})^2} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot e(t) \ . \tag{4.39}$$

A partir do modelo (4.39), pode-se perceber que os modelos SISO relacionando a

entrada γ com a orientação do veículo θ e a coordenada local y_L possuem a mesma dinâmica, porém, com ganhos estáticos diferentes.

Com o objetivo de calcular a seqüência de controles futuros $\gamma(k)$, $\gamma(k+1)$, $\cdots, \gamma(k+N_u-1)$ é necessário obter as predições das saídas $\hat{\mathbf{y}}(k+j|k)$, com $j = 1, \cdots, N_2$. Neste caso as saídas $\mathbf{y}(k)$ do modelo são $y_L(k)$ e $\theta(k)$. Usando o modelo CARIMA (4.39) para o robô, a equação Diofantina (4.26) é resolvida para obter a predição ótima de cada saída, tendo em vista a estrutura desacoplada do modelo (NORMEY-RICO *et al.*, 1999). Desta forma, as expressões da predição ótima para as variáveis $y_L(k)$ e $\theta(k)$ são:

$$\hat{\theta}(k+j|k) = 2\cdot\hat{\theta}(k+j-1|k) - \hat{\theta}(k+j-2|k) + v(k-1)\cdot T\cdot\Delta\gamma(k+j-1) \quad (4.40)$$

$$\hat{y}_L(k+j|k) = 2\cdot\hat{y}_L(k+j-1|k) - \hat{y}_L(k+j-2|k) + \frac{(v(k-1)\cdot T)^2}{2}\cdot\Delta\gamma(k+j-1)\ , \quad (4.41)$$

ou em forma vetorial:

$$\begin{bmatrix} \hat{\theta}(k+1|k) \\ \vdots \\ \hat{\theta}(k+N_{2\theta}|k) \\ \hat{y}_L(k+1|k) \\ \vdots \\ \hat{\theta}(k+N_{2y_L}|k) \end{bmatrix} = \mathbf{G}\cdot \begin{bmatrix} \Delta\gamma(k) \\ \Delta\gamma(k+1) \\ \\ \Delta\gamma(k+N_u-1) \end{bmatrix} + \begin{bmatrix} f_\theta(1) \\ \vdots \\ f_\theta(N_{2\theta}) \\ f_{y_L}(1) \\ \vdots \\ f_{y_L}(N_{2y_L}) \end{bmatrix}\ , \quad (4.42)$$

onde as matrizes $\mathbf{G}$ e $\mathbf{f}$, devido a característica integradora do modelo cinemático do veículo em coordenadas locais, são definidas da seguinte maneira:

$$\mathbf{G} = \begin{bmatrix} v(k-1)\cdot T\cdot G_1 \\ \dfrac{(v(k-1)\cdot T)^2}{2}\cdot G_2 \end{bmatrix},\ \mathbf{G}_p = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 2 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ N_{2p} & N_{2p}-1 & \cdots & N_{2p}-N_u-1 \end{bmatrix}, \tag{4.43}$$

$$\mathbf{f} = \begin{bmatrix} f_1 & 0 \\ 0 & f_2 \end{bmatrix} \cdot \begin{bmatrix} \hat{\theta}\,(k|k) \\ \hat{\theta}\,(k-1|k) \\ \hat{y}_L\,(k|k) \\ \hat{y}_L\,(k-1|k) \end{bmatrix},\ \mathbf{f}_p = \begin{bmatrix} 2 & -1 \\ 3 & -2 \\ \vdots & \vdots \\ N_{2p}+1 & -N_{2p} \end{bmatrix}. \tag{4.44}$$

Portanto, com a expressão da predição das saídas (4.42), minimiza-se a função custo (4.35) em relação a $\Delta\gamma$, e a partir da expressão (4.37), o cálculo da lei de controle linear é dado por:

$$\Delta\gamma(k) = \mathbf{K}\cdot(\mathbf{y}_{ref} - \mathbf{f}\,), \tag{4.45}$$

com

$$\mathbf{y}_{ref} = \begin{bmatrix} \theta_{ref}(k+1) \\ \vdots \\ \theta_{ref}(k+N_{2\theta}) \\ y_{Lref}(k+1) \\ \vdots \\ y_{Lref}(k+N_{2y_L}) \end{bmatrix}.$$

Como a matriz G_p de resposta ao degrau é proporcional ao ganho estático do sistema (g_{ep}), para normalizar o efeito da ponderação $\mathbf{R}$ na matriz $\mathbf{K}$ é necessário utilizar um valor proporcional a g_e^2. Assim, escolhe-se $\mathbf{R} = \mathbf{I}\cdot R_e\cdot g_e^2$ e varia-se R_e para ponderar o sinal de controle (RAFFO, 2005). Como o sistema possui duas saídas e uma entrada de

controle, optou-se por escolher o ganho estático da saída que apresenta maior influência no controle. No caso do modelo cinemático em coordenadas locais, escolhe-se o ganho do estado θ e a ponderação $\mathbf{R}$ é escrita da seguinte forma:

$$\mathbf{R} = \mathbf{I} \cdot R_e \cdot (v(k) \cdot T)^2 \,, \tag{4.46}$$

onde $\mathbf{I}$ é uma matriz identidade de dimensão igual à soma dos horizontes de controle.

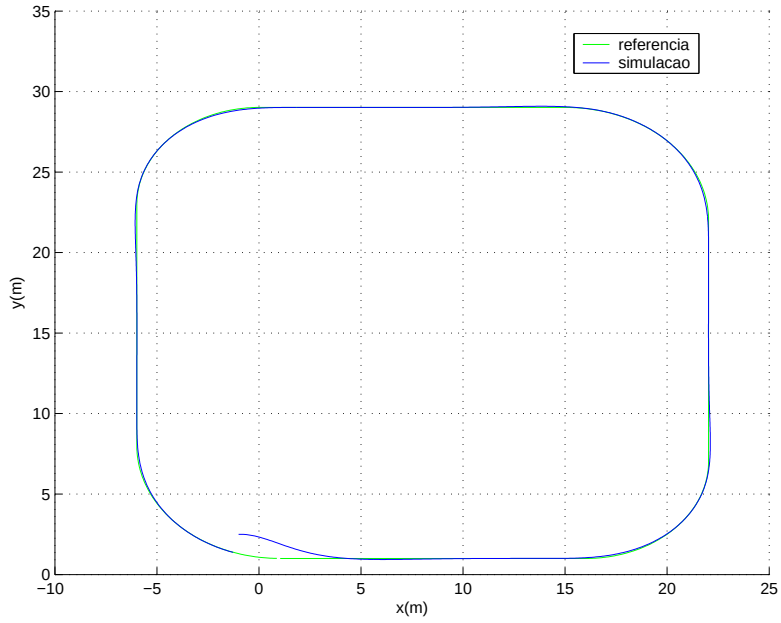

Figura 4.12: Simulação da trajetória no plano xy com GPC aplicado ao modelo cinemático em robo diferencial.

Na Figura 4.12 e na Figura 4.13 são apresentados resultados da simulação do GPC aplicado a um veículo diferencial, utilizando os parâmetros apresentados pela Tabela 4.1. A simulação foi realizada com $N_{2\theta} = N_{2y_L} = N_u = N$ e considerando a velocidade v_D constante, de modo que o ganho da planta se mantém constante durante todo o percurso. A partir dos resultados obtidos na simulação, pode-se observar que o robô segue o caminho determinado de maneira satisfatória. A técnica *pure pursuit* de geração de trajetória de aproximação, representada pelo bloco *Gerador e Supervisor da Trajetória de Referência* da estrutura da Figura 4.9, demonstra-se muito útil para evitar grandes variações no ângulo de orientação θ, proporcionando uma aproximação suave até o caminho de referência. Através da utilização do *look-ahead* adaptativo, a

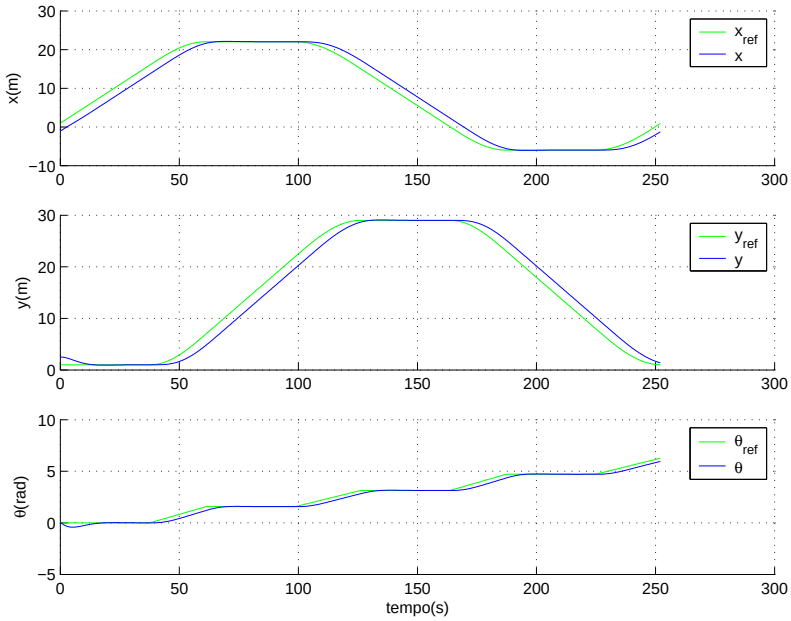

Figura 4.13: Simulação da evolução temporal dos estados x, y e θ com GPC aplicado ao modelo cinemático em robo diferencial.

distância entre a posição atual do robô e o ponto desejado sobre a trajetória é monitorada periodicamente e serve como base para a redefinição do *look-ahead*. Se a distância do robô ao ponto desejado é maior que o valor mínimo, o *look-ahead* é redefinido, sendo igualado à distância atual do robô até o ponto desejado. Se o robô se encontra sobre o ponto desejado, é assegurado um valor mínimo para garantir a estabilidade e evitar oscilações. O *valor mínimo* do *look-ahead* é determinado a partir das dimensões do veículo e da velocidade v, sendo, no caso da simulação, utilizado o valor correspondente ao comprimento do veículo. O monitoramento da distância entre a posição atual do robô e o ponto desejado é aqui realizada a cada dez amostras.

Os parâmetros utilizados para simulação e apresentados na Tabela 4.1 foram escolhidos de modo a se obter um percurso fiel à trajetória de referência. Outras simulações realizadas mostraram que a utilização de horizontes mais elevados em casos de posição inicial fora da trajetória implica a realização do percurso pelo lado de dentro das curvas, como se o robô procurasse um atalho até o fim da curva. Isso se deve ao fato de que, como o robô inicia o percurso distante da trajetória, um horizonte elevado faz com que ele enxergue pontos da trajetória de referência muito à frente da sua posição atual. Além disso, o robô sempre estará distante do ponto desejado para aquele instante, o

Parâmetro	**Valor Utilizado**
período de amostragem T	$0.2s$
velocidade linear v	$0.2m/s$
N (horizontes)	20
Q_θ (peso de θ)	1
Q_{y_L} (peso de y_L)	1
R_e (peso do controle)	$62,5$
α (look-ahead)	adaptativo

Tabela 4.1: Parâmetros de sintonia do controlador na simulação com o GPC aplicado ao modelo cinemático em robô diferencial.

que mantém o *look-ahead* elevado, também contribuindo para que o robô corte, pelo lado de dentro, os cantos das trajetórias. Em outras palavras, tem-se que como a velocidade é mantida constante, o único jeito de se atingir o ponto desejado partindo fora da trajetória é percorrendo o caminho mais curto.

Modelo Cinemático Direcional

A aplicação do GPC ao modelo cinemático de robôs direcionais do tipo automóvel é muito semelhante à aplicação em modelos diferenciais, à medida que ambos os modelos linearizados em coordenadas locais apresentam a mesma dinâmica e diferem entre si apenas nos ganhos estáticos. A seguir, tem-se novamente o modelo cinemático linearizado em coordenadas locais para robôs direcionais, dado pela expressão 4.8.

$$\begin{cases} x_L(k+1) = x_L(k) + v_D(k) \cdot T \\ y_L(k+1) = y_L(k) + v_D(k) \cdot T \cdot \delta_D(k) \\ \theta(k+1) = \theta(k) + \dfrac{v_D(k) \cdot T}{d_D} \cdot \delta_D(k) \end{cases} .$$

A função de transferência que relaciona as saídas θ e y_L com a entrada δ_D é dada por:

$$\left(1 - z^{-1}\right) \cdot \begin{bmatrix} y_L(k) \\ \theta(k) \end{bmatrix} = v_D(k-1) \cdot T \cdot \begin{bmatrix} 1 \\ 1/d_D \end{bmatrix} \cdot \delta_D(k-1) \ , \tag{4.47}$$

com v_D considerada constante entre duas amostras.

O modelo CARIMA (4.24) para o modelo (4.47) é dado por:

$$\begin{bmatrix} y_L(k) \\ \theta(k) \end{bmatrix} = \frac{v_D(k-1) \cdot T}{(1 - z^{-1})} \cdot \begin{bmatrix} 1 \\ 1/d_D \end{bmatrix} \cdot \delta_D(k-1) + \frac{1}{(1 - z^{-1})^2} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot e(t) \ . \tag{4.48}$$

Da mesma forma que o modelo diferencial dado pela expressão (4.39), o modelo CARIMA para robôs direcionais apresenta a mesma dinâmica para as variáveis de saída θ e y_L, porém com ganhos estáticos diferentes. Resolvendo-se a equação Diofantina (4.26) de modo a se obter a predição ótima para o modelo direcional, chega-se às seguintes expressões para $\hat{\theta}(k+j|k)$ e $\hat{y}(k+j|t)$:

$$\hat{\theta}(k+j|k) = 2 \cdot \hat{\theta}(k+j-1|k) - \hat{\theta}(k+j-2|k) + \frac{v_D(k-1) \cdot T}{d_D} \cdot \Delta\delta_D(k+j-1) \ , \tag{4.49}$$

$$\hat{y}_L(k+j|k) = 2 \cdot \hat{y}_L(k+j-1|k) - \hat{y}_L(k+j-2|k) + v_D(k-1) \cdot T \cdot \Delta\delta_D(k+j-1) \ , \tag{4.50}$$

ou em forma vetorial:

$$\begin{bmatrix} \hat{\theta}(k+1|k) \\ \vdots \\ \hat{\theta}(k+N_{2\theta}|k) \\ \hat{y}_L(k+1|k) \\ \vdots \\ \hat{\theta}(k+N_{2y_L}|k) \end{bmatrix} = \mathbf{G} \cdot \begin{bmatrix} \Delta\delta_D(k) \\ \Delta\delta_D(k+1) \\ \\ \Delta\delta_D(k+N_u-1) \end{bmatrix} + \begin{bmatrix} f_\theta(1) \\ \vdots \\ f_\theta(N_{2\theta}) \\ f_{y_L}(1) \\ \vdots \\ f_{y_L}(N_{2y_L}) \end{bmatrix} \ , \tag{4.51}$$

onde a matriz **f** é a mesma da expressão (4.44) e a matriz **G** é dada por:

$$\mathbf{G} = \begin{bmatrix} v_D(k-1) \cdot T \cdot G_1 \\ \dfrac{(v_D(k-1) \cdot T)^2}{2} \cdot G_2 \end{bmatrix} \tag{4.52}$$

sendo as matrizes G_1 e G_2 as mesmas da expressão (4.43).

Por fim, o cálculo da ação de controle $\Delta\delta_D$ é obtido por:

$$\Delta\delta_D(k) = \mathbf{K} \cdot (\mathbf{y}_{ref} - \mathbf{f}) \ , \tag{4.53}$$

sendo y_{ref} igual ao da expressão (4.45).

Da mesma forma como no caso do modelo diferencial, a ponderação do controle é feita com $\mathbf{R} = \mathbf{I} \cdot R_e \cdot g_e^2$, sendo escolhido o ganho estático da saída θ. Assim, a ponderação **R** para o modelo cinemático direcional é dada por:

$$\mathbf{R} = \mathbf{I} \cdot R_e \cdot \left(\frac{v_D(k) \cdot T}{d_D} \right)^2 \ , \tag{4.54}$$

onde novamente **I** é uma matriz identidade de dimensão igual à soma dos horizontes de controle.

Os resultados da simulação do GPC aplicado ao modelo cinemático de um robô direcional são apresentados na Figura 4.14 e na Figura 4.15. Novamente a simulação é realizada com $N_{2\theta} = N_{2y_L} = N_u = N$, embora outras configurações também possam ser usadas. Também aqui a velocidade v_D é considerada constante.

Os parâmetros utilizados na simulação com veículo direcional são apresentados na Tabela 4.2.

A partir dos resultados obtidos na simulação para o robô direcional, pode-se observar que, assim como no caso diferencial, o robô segue o caminho determinado com bom desempenho. As observações relacionadas à técnica *pure pursuit* mostram que a influência desta no desempenho e estabilidade do sistema ocorre de maneira similar

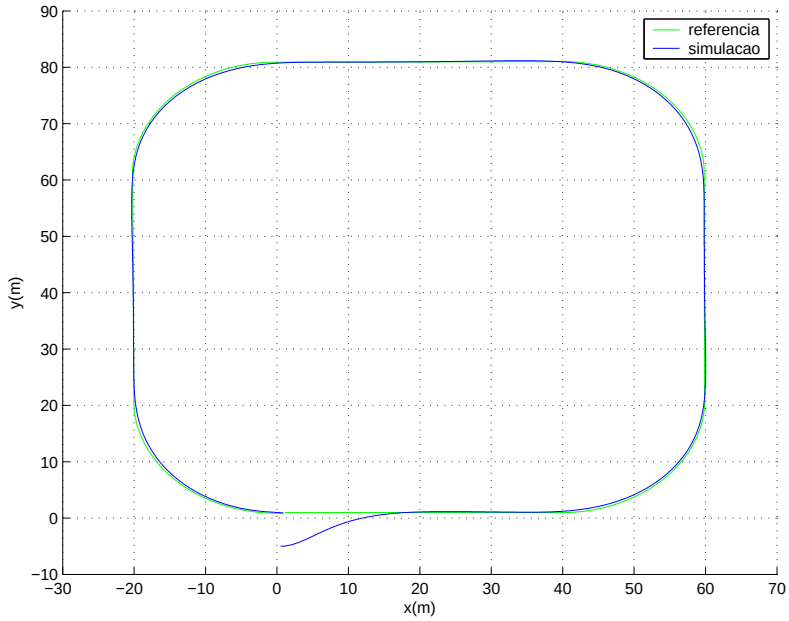

Figura 4.14: Simulação da trajetória no plano xy com GPC aplicado ao modelo cinemático em robô direcional.

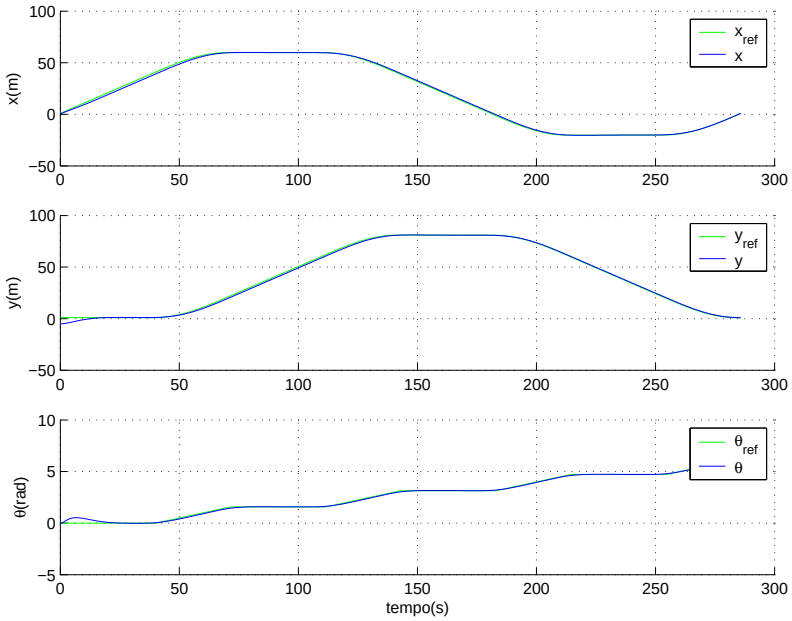

Figura 4.15: Simulação da evolução temporal dos estados x, y e θ com GPC aplicado ao modelo cinemático em robô direcional.

ao caso diferencial. A utilização de trajetórias suaves de aproximação em robôs direcionais é importante, não só para evitar variações elevadas no ângulo θ, mas também para evitar valores elevados na ação de controle dada pelo ângulo δ_D, a fim de manter a validade do modelo linearizado dado pela expressão (4.8) e respeitar os limites físicos

Parâmetro	**Valor Utilizado**
período de amostragem T	$0.1s$
velocidade linear v_D	$1m/s$
N (horizontes)	30
Q_θ (peso de θ)	1
Q_{y_L} (peso de y_L)	1
R_e (peso do controle)	21240
α (look-ahead)	adaptativo

Tabela 4.2: Parâmetros de sintonia do controlador na simulação com o GPC aplicado ao modelo cinemático em robô direcional.

de robôs desta natureza. Da mesma forma que no caso diferencial, percursos com velocidade constante e posições iniciais fora da trajetória de referência implicam *look-ahead* elevado e, como conseqüência, erros de seguimento ao longo das curvas. Com relação a escolha dos horizontes, se verifica também que, assim como no caso diferencial, horizontes de predição elevados fazem com que o robô tenda a antecipar a realização das curvas, percorrendo-as pelo lado interno.

A escolha dos parâmetros utilizados na simulação (Tabela 4.2) novamente foi feita considerando as condições de operação e as dimensões do veículo, de modo a se obter um percurso fiel à trajetória dentro dos níveis de precisão desejados. Em função da velocidade utilizada nas simulações do modelo direcional ser maior que a utilizada no modelo diferencial, foram escolhidos horizontes mais elevados, para permitir que o robô reaja com antecedência à chegada das curvas, porém tomando cuidado para que não ocorram erros no percurso destas, devido aos fatores explicados anteriormente. Assim como no caso das simulações com robô diferencial, o bloco *Gerador e Supervisor da Trajetória de Referência* monitora, a cada dez amostras, a distância entre a posição atual do robô e o ponto desejado, redefinindo quando necessário, o valor do *look-ahead* com base nesta distância e no comprimento do robô.

4.6.3 GPC Aplicado ao Modelo Dinâmico

Conforme descrito no início desta Seção, para realizar com bom desempenho e de forma adequada o controle de seguimento de trajetória em robôs com quantidade significativa de massa e em velocidades elevadas, torna-se necessária a consideração do comportamento dinâmico. Neste são descritas as forças envolvidas na produção do deslocamento e a suas influências sobre as condições do robô. Também como descrito anteriormente nesta Seção, o controle do modelo dinâmico é desenvolvido para robôs direcionais, com a finalidade de implementação em veículos do tipo automóvel.

Para a aplicação do algoritmo GPC ao controle da dinâmica de robôs direcionais, parte-se das funções de transferência que representam o comportamento dinâmico de robôs direcionais. Em RAFFO (2005), foi obtida, a partir das expressões (4.9, 4.10), a função de transferência correspondente às dinâmicas da orientação do vetor velocidade β e da velocidade angular do veículo $\dot{\theta}$, considerando-se as condições iniciais nulas. Esta função de transferência é dada pela expressão (4.55)

$$G(s)_{\beta,\,\theta} = \frac{1}{s^2 + (a+d)\cdot s + (ad - bc)} \begin{bmatrix} e\cdot s + (de + bf) \\ f\cdot s + (af + ce) \end{bmatrix} \tag{4.55}$$

com:

$$a = \frac{c_T + c_D}{m\cdot v}\,, \quad b = \frac{c_T\cdot d_T - c_D\cdot d_D}{m\cdot v^2} - 1\,, \quad c = \frac{c_T\cdot d_T - c_D\cdot d_D}{J_z}\,,$$

$$d = \frac{c_T\cdot d_T^2 + c_D\cdot d_D^2}{J_z\cdot v}\,, \quad e = \frac{c_T\cdot d_T^2 + c_D\cdot d_D^2}{J_z\cdot v}\ \mathsf{e}\,, \quad f = \frac{d_D\cdot(c_D + F_{xD})}{J_z}\,.$$

Quanto à dinâmica da velocidade, em RAFFO (2005) é utilizada a expressão (4.11), obtida *in locus* em GOMES (2003) já na forma de função de transferência. Com isso, utilizando-se ferramenta computacional, foram obtidas as funções de transferências em tempo discreto equivalentes ao modelo dinâmico dado pelas expressões (4.55, 4.11), de modo a compor os polinômios $A_p(z^{-1})$ e $B_{pq}(z^{-1})$ do modelo CARIMA (4.24). O algoritmo GPC é executado nas duas malhas de controle do modelo dinâmico do robô,

seguindo os passos da expressão (4.25) até (4.37), onde se obtêm as leis de controle linear para os dois modelos:

$$\Delta\delta_D(k) = \mathbf{K} \cdot (\mathbf{y}_{ref} - \mathbf{f}) , \quad \mathbf{y}_{ref} = \begin{bmatrix} \beta_{ref}(k+1) \\ \vdots \\ \beta_{ref}(k+N_{2\beta}) \\ \dot{\theta}_{ref}(k+1) \\ \vdots \\ \dot{\theta}_{ref}(k+N_{2\dot{\theta}}) \end{bmatrix} \tag{4.56}$$

e

$$\Delta\varphi(k) = \mathbf{K} \cdot (\mathbf{v}_{ref} - \mathbf{f}) , \quad \mathbf{v}_{ref} = \begin{bmatrix} v_{ref}(k+1) \\ \vdots \\ v_{ref}(k+N_{2v}) \end{bmatrix} \tag{4.57}$$

Como já foi mencionado na Seção 4.3, a dinâmica da velocidade descrita pelo modelo da expressão (4.11) é válida apenas para situações de aceleração, de modo que o controlador da dinâmica da velocidade que utiliza este modelo, atua apenas no primeiro quadrante, ou seja, atua apenas nos casos em que a velocidade de referência é maior que a velocidade medida. Em RAFFO (2005), a situação em que a velocidade de referência é menor que a velocidade medida não é tratada, de modo que o controlador não atua sobre o sistema de freios, o que, neste caso, deve ser tratado separadamente. A Figura 4.16 apresenta a estrutura das duas malhas de controle da dinâmica de robôs direcionais utilizando o algoritmo GPC.

Nesta estrutura, as referências futuras para ambas as malhas de controle são geradas pelo bloco *Gerador e Supervisor da Trajetória de Referência* através dos valores futuros de δ_D e v. Os valores futuros da velocidade ($v_{ref}(k+j)$) são passados diretamente para a malha da dinâmica da velocidade. Já a malha da dinâmica de β e $\dot{\theta}$ deve receber, como referência, os valores de $\beta_{ref}(k+j)$ e $\dot{\theta}_{ref}(k+j)$, que são obtidos a partir dos valores de $\delta_{Dref}(k+j)$ por meio do bloco *Relação Trigonométrica* que define a relação estática entre o ângulo das rodas dianteiras δ_D e o ângulo β da velocidade do CM (RAFFO, 2005). As grandezas não mensuráveis de β e v necessárias à predição dos

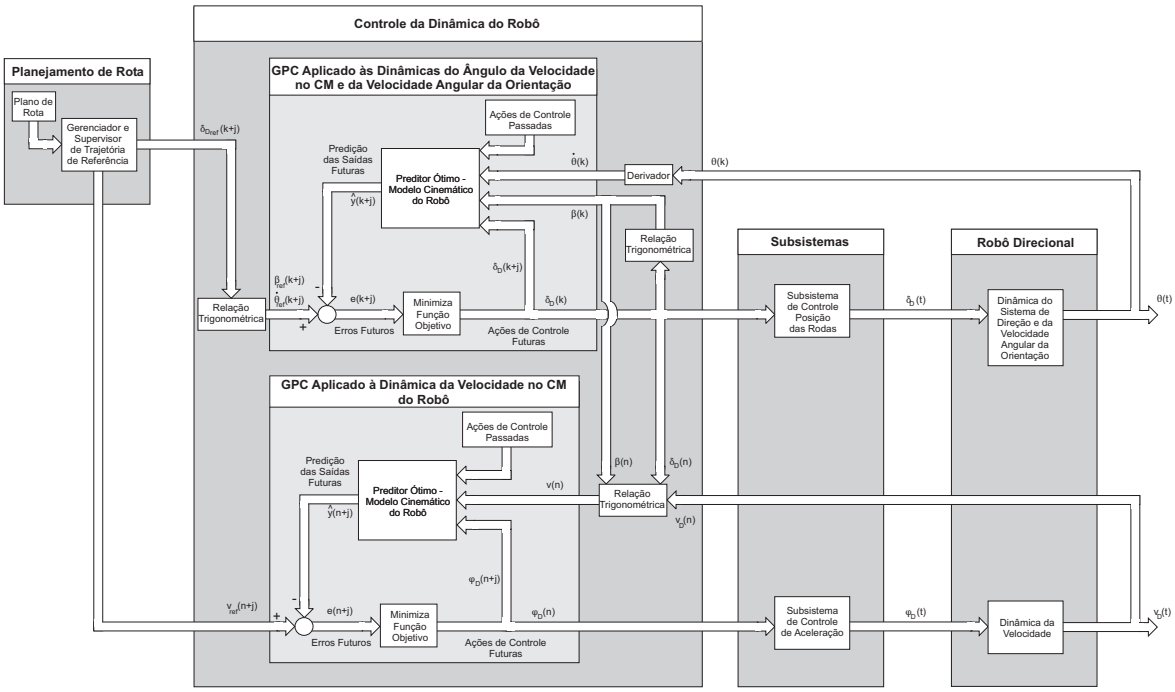

Figura 4.16: GPC aplicado ao modelo dinâmico.

controladores são obtidas também por meio do bloco *Relação Trigonométrica*, a partir das medições de θ e v_D. O valor de $\dot{\theta}$, também necessário à predição do controlador do ângulo da velocidade e do próprio $\dot{\theta}$, é aqui obtido derivando-se o valor medido de θ.

Porém, o controle de seguimento de trajetória utilizando modelo dinâmico pode apresentar erro em regime permanente para as variáveis β e $\dot{\theta}$. Para funcionar corretamente, as referências de β_{ref} e $\dot{\theta}_{ref}$ devem ser geradas pelo controlador da cinemática do robô de forma suave, validando o modelo linearizado (RAFFO, 2005).

4.6.4 Controle em Cascata para Seguimento de Trajetória de Veículos Direcionais

Uma solução para a obtenção de resultados satisfatórios em aplicações de alto desempenho, evitando a presença dos erros em regime permanente que ocorrem no controle baseado apenas em modelo dinâmico, consiste na utilização de uma estrutura em cascata com controladores de cinemática e dinâmica. Com isso, são utilizados todos os níveis apresentados na Figura 2.1.

Utilizando-se o algoritmo GPC aplicado ao problema de seguimento de trajetória,

em RAFFO (2005) é proposta uma estrutura em cascata para controlar a cinemática e a dinâmica do veículo, conforme mostra a Figura 4.17.

Esta estrutura utiliza os quatro níveis apresentados na Figura 2.1. O primeiro nível é composto pelos subsistemas do veículo, responsáveis pelas funções de frenagem, aceleração e direção. Estes subsistemas são responsáveis por realizar a aplicação dos comandos δ_D e φ gerados pelo segundo nível, onde estão presentes os GPC's responsáveis pelo controle das variáveis β, $\dot{\theta}$ e v que descrevem o comportamento dinâmico. Novamente aqui a situação de frenagem deve ter o seu comportamento dinâmico tratado separadamente, pois a dinâmica do sistema de frenagem não é considerada. No terceiro nível é realizado o controle da cinemática, no qual a seqüência de controles futuros calculada pelo controlador GPC baseado no modelo cinemático em coordenadas locais é passada, via bloco *Relações Trigonométricas*, como referência futura para o controlador da dinâmica. No quarto e último nível, estão presentes o planejador de trajetória (bloco *Plano de Rota*) e o *Gerador e Supervisor da Trajetória de Referência*. O *Plano de Rota* é encarregado de definir o caminho em coordenadas globais a ser seguido e passá-lo para o *Gerador e Supervisor da Trajetória de Referência*, que, por sua vez, gera a cada amostra, através da técnica *pure pursuit*, a trajetória de aproximação no sistema de coordenadas locais fixo ao robô no instante k, como referência futura para o controlador da cinemática, e os valores de velocidade futuros de referência para o controlador da dinâmica da velocidade via realimentação *feed forward*. A cada dez amostras, o *Gerador e Supervisor da Trajetória de Referência* realiza também a verificação da distância entre o veículo e o ponto de destino sobre a trajetória de referência e, quando necessário, redefine o parâmetro *look-ahead*.

Maiores detalhes sobre a estrutura em cascata da Figura 4.17 são apresentados em RAFFO (2005).

Uma estrutura alternativa, na qual se utiliza o controlador ACC proposto em GOMES (2003) para tratar da dinâmica da velocidade no centro de massa, é apresentada na Figura 4.18.

Através da utilização do controlador ACC, o controle da velocidade é realizado, tanto nas situações de aceleração quanto nas situações de frenagem, através da estrutura chaveada composta por dois controladores PID que consideram as dinâmicas

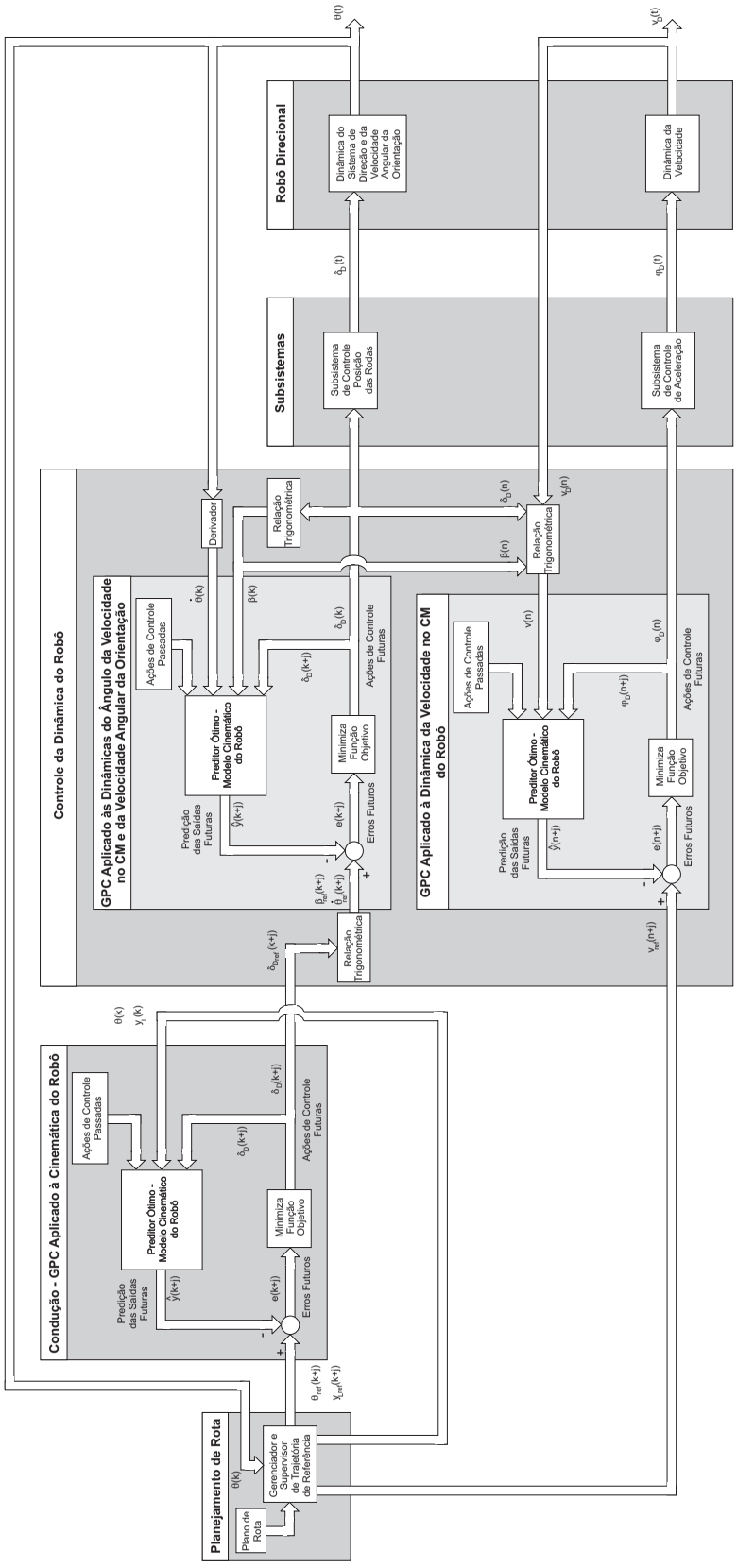

Figura 4.17: Controle em cascata com GPC aplicado aos modelos dinâmico e cinemático (RAFFO, 2005).

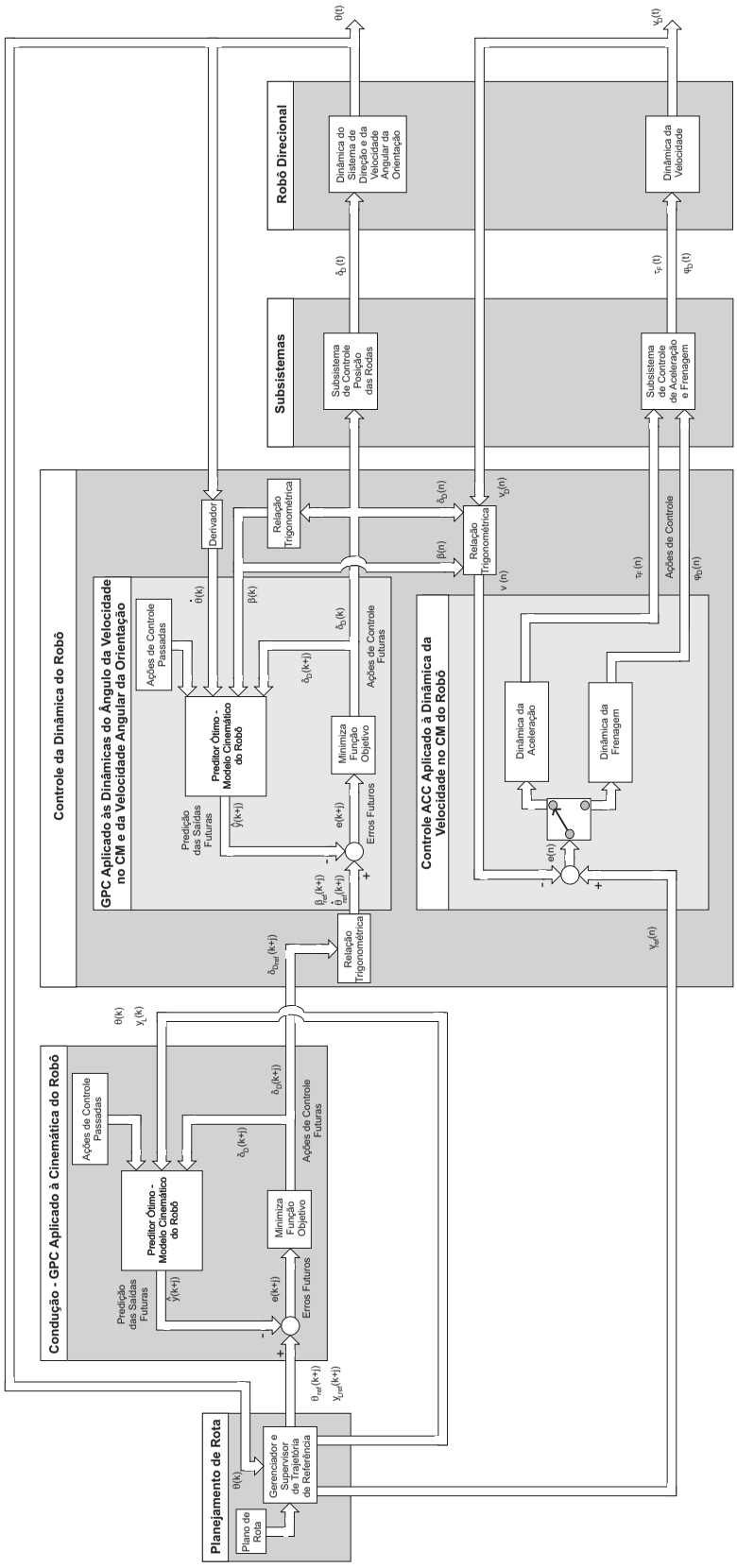

Figura 4.18: Controle em cascata com GPC aplicado ao modelo cinemático e ao modelos dinâmicos de β e $\dot{\theta}$ e com controlador ACC aplicado ao modelo dinâmico de v.

diferenciadas das situações de aceleração e frenagem, descritas pelas expressões (4.11, 4.12). A utilização da estrutura chaveada permite a condução estável nas situações de subida e descida de planos inclinados. Maiores detalhes sobre o controlador ACC são apresentados em GOMES (2003).

4.7 Conclusões

Este capítulo apresentou a modelagem da duas estruturas de robôs móveis propostas para a aplicação do sistema de controle de seguimento de trajetória, bem como a apresentação da técnica de controle preditivo escolhida para ser utilizada no projeto. São analisados também algoritmos de geração de trajetória de aproximação e conversão de coordenadas, necessários à técnica de controle utilizada.

A opção por técnica de CPBM linear é justificada pelo fato de que técnicas não lineares de controle preditivo apresentam alto custo computacional e inviabilizam a aplicação em tempo-real para sistemas embarcados, utilizando as plataformas disponíveis. Assim, optou-se pela utilização do algoritmo de CPBM linear denominado GPC, por apresentar custo computacional menor se comparado a outras técnicas lineares aplicáveis, como o controle baseado em espaço de estados analisado em RAFFO (2005). A utilização de técnica de controle linear implica a necessidade de linearização dos modelos dos veículos a serem controlados, de maneira que alguns cuidados devem ser tomados para garantir as condições que tornam estes modelos válidos.

Os modelos utilizados para representar a cinemática dos robôs são linearizados sob as considerações de que a variação do ângulo de orientação e do ângulo da posição das rodas (no caso de robôs direcionais) é pequena a cada período de amostragem do controlador. Porém, em casos específicos em que o robô esteja situado fora da trajetória de referência, a tendência do controlador é aplicar grandes esforços de controle para corrigir o erro elevado, o que tende a provocar grandes variações angulares que acabam por invalidar o modelo linearizado. Daí surge a necessidade da geração de trajetórias que conduzam o robô da sua posição atual até o ponto desejado sobre a trajetória de referência, de modo que, do ponto de vista do controlador, o robô se encontre sempre posicionado sobre a trajetória desejada.

Para se obter estas trajetórias de aproximação, foi utilizada a técnica *pure pursuit* que gera uma seqüência de arcos produzindo uma trajetória suave que liga a posição atual do robô ao ponto desejado sobre a trajetória. Através das simulações realizadas se verificou que a técnica é eficiente para aproximar o robô até a trajetória de referência, garantindo a validade do modelo linearizado utilizado pelo controlador. As simulações mostraram também a necessidade de um ajuste adequado do parâmetro *look-ahead* para obter um compromisso entre estabilidade e fidelidade no percurso da trajetória, o que pode ser alcançado através da técnica de *look-ahead* adaptativo.

Ainda com a finalidade de se analisar a aplicação a robôs direcionais em altas velocidades, foi apresentada a modelagem dinâmica destes, onde estuda-se a forma de atuação e a influência das forças envolvidas no processo de condução. Com a obtenção, sob certas condições de contorno, do modelo linearizado do comportamento dinâmico, foram propostas duas estruturas de controle em cascata para controlar o comportamento dinâmico e cinemático, com base em RAFFO (2005) e GOMES (2003).

No próximo capítulo serão apresentados os conceitos empregados em computação em tempo-real orientada a objetos que são utilizados para modelar o sistema de controle de seguimento de trajetória.

Capítulo 5

Computação Tempo-Real Orientada a Objetos

5.1 Introdução

Os processos automatizados de produção industrial possuem uma complexidade bastante elevada, refletindo diretamente na complexidade dos sistemas que os controlam. Neste tipo de aplicação, a exemplo do SCST descrito neste trabalho, é muito importante a definição de uma boa arquitetura. Aspectos como modularidade, coesão e acoplamento têm grande influência sobre os custos de instalação, manutenção e engenharia. Através da adoção do paradigma de orientação a objetos, obtêm-se sistemas com propriedades desejáveis em relação a tais aspectos (BECKER, 2003). Estes sistemas são organizados através de objetos, que representam as entidades do domínio do problema. As chamadas classes definem objetos com características e comportamentos comuns, e podem ser organizadas em estruturas hierárquicas através de conceitos de herança. Como resultado, obtêm-se softwares mais fáceis de entender, corrigir e modificar, facilitando muito o trabalho em equipes de desenvolvimento.

Além disso, objetos são muito adequados à programação concorrente, pois sua autonomia lógica os tornam unidades naturais para execução concorrente (BECKER e PEREIRA, 2002). Assim, o paradigma de orientação a objeto consiste em um método adequado para representar processos concorrentes presentes no mundo real de uma maneira natural e de fácil compreensão.

A adoção do paradigma de OO remete preferencialmente à utilização da *Unified Modeling Language* (UML) (RUMBAUGH, 1991), definida pelo *Object Management Group*

(OMG) como notação padronizada para modelagem destes sistemas. No contexto deste trabalho é importante destacar a existência do perfil UML-TR, também padronizado pelo OMG, que permite decorar os diagramas UML com diversos tipos de restrições temporais presentes em sistemas de tempo-real.

O restante deste capítulo tem a finalidade de apresentar a metodologia de análise e projeto orientado a objetos (OO) baseada em UML, utilizada para o desenvolvimento do sistema de controle de seguimento de trajetória, (SCST), bem como os principais conceitos de computação em tempo-real utilizados para descrever os requisitos temporais do sistema em questão.

5.2 Projeto Orientado a Objetos com UML

A UML é uma notação gráfica padronizada que permite, por meio de diagramas, representar as características dos projetos OO. Ela unificou os diversos esquemas de notação que existiam em meados da década de 90, sendo em 1997 foi definida pelo OMG como a primeira notação padrão para modelagem OO. A partir daí, o OMG assumiu a responsabilidade pela manutenção e revisão continuadas da UML. Com base na utilização da UML, a seguir serão descritas em ordem cronológica, as etapas de um projeto orientado a objeto baseado na metodologia proposta por DEITEL e DEITEL (2001).

5.2.1 Definição do Problema

Nesta etapa é realizada a definição detalhada do problema, que resulta em elencar a relação de requisitos do sistema. Esta relação serve para orientar a equipe de desenvolvimento e evitar ocasionais mudanças durante a etapa de desenvolvimento, devido a erros de planejamento.

A linguagem UML oferece o *diagrama de casos de usos* para representar os requisitos do sistema a ser desenvolvido. Os diagramas de casos de uso modelam as interações entre os clientes externos e os *casos de uso* do sistema. Cada caso de uso representa uma funcionalidade diferente que o sistema oferece. Em um diagrama de casos de uso, quaisquer entidades externas que interagem com o sistema, tais como pessoas, robôs ou mesmo outros sistemas, são chamadas de *atores* e são representados por bonecos, como

mostra a Figura 5.1. Casos de uso são modelados por elipses dentro de um contorno retangular, ou *caixa do sistema* que representa o sistema propriamente dito.

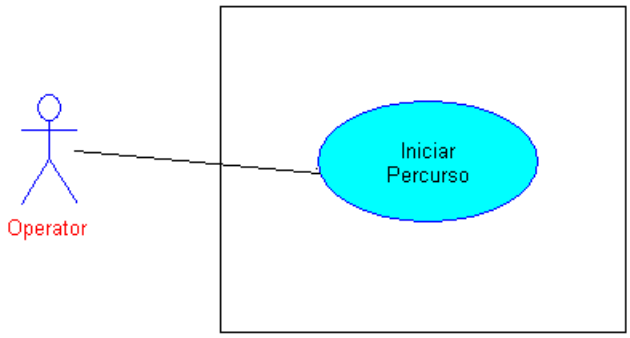

Figura 5.1: Exemplo de diagrama de caso de uso.

5.2.2 Modelagem das Classes

A etapa seguinte consiste na definição das classes do sistema e na forma como elas deverão compor o sistema. As classes e seu relacionamentos são modeladas em UML através de *diagramas de classes*, onde estas são representadas por um retângulo dividido em três partes. A parte superior contém o nome da classe, a parte do meio contém os atributos e a parte inferior contém as operações da classe. Atributos de uma classe são dados de tipos primitivos ou tipos criados pelo usuário, como outras classes. Já as operações são funções membros da classe que tem acesso a todos os atributos desta. A Figura 5.2 apresenta um exemplo da diagrama de classes.

Classes se relacionam umas com as outras através de *associações*, representadas por um linha cheia que conecta duas classes. Associações *unidirecionais* apresentam um seta em uma das extremidades indicando a direção da associação. Já associações *bidirecionais* não apresentam setas. Os números junto às linhas de associação expressam valores de *multiplicidade*, e indicam quantos objetos de uma classe participam das associações. Na Figura 5.2, por exemplo, um objeto da classe *OperatorIO* se associa com dois objetos da classe *Button* e um objeto das classes *Reference*, *Display* e *Keyboard*. A classe *OperatorIO* tem uma associação do tipo *um-para-dois* com a classe *Button* e uma associação *um-para-um* com as classes *Reference*, *Display* e *Keyboard*.

O losango cheio preso às linhas de associação da classe *OperatorIO* indica que esta

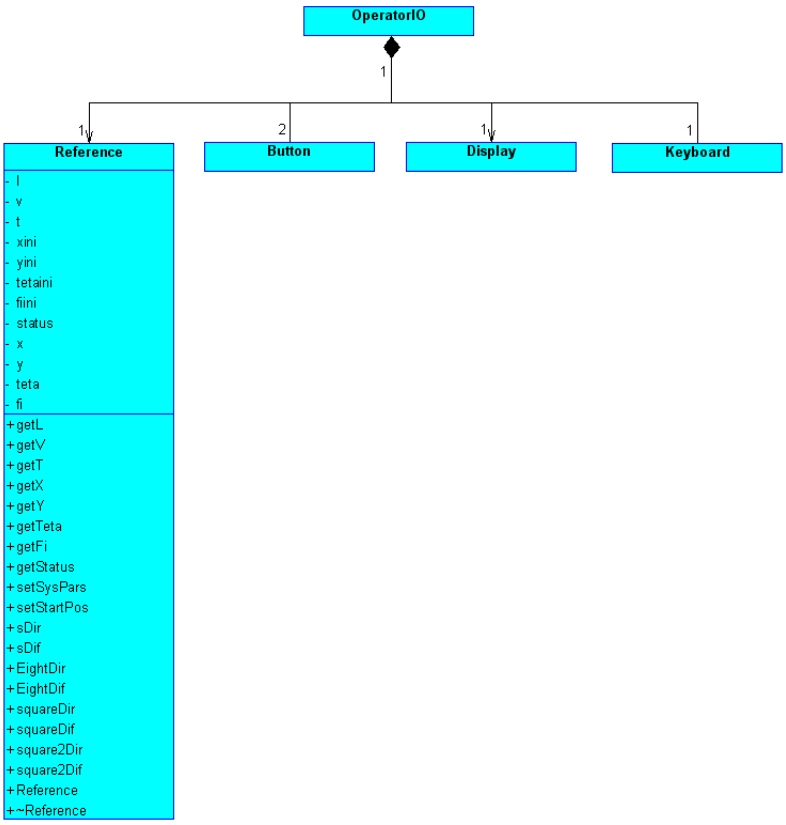

Figura 5.2: Exemplo de diagrama de classes.

classe possui um relacionamento de composição com as classes *Reference*, *Display* e *Keyboard*. A composição indica um relacionamento todo/parte. A classe que possui o losango cheio em sua extremidade da linha é o todo e a classe na outra extremidade é a parte, de modo que o todo é composto pelas partes. Como mostra ainda o exemplo da Figura 5.2, a UML permite o truncamento da descrição dos atributos e operações de modo a tornar legíveis os diagramas.

Para modelar o conceito de *herança*, onde classes novas são criadas a partir de classes existentes pela absorção de seus atributos e operações, é utilizado o símbolo de um triângulo em uma extremidade de associação, como mostra a Figura 5.3. A classe com o triângulo na sua extremidade da linha é denominada *classe base*, e as classes da outra extremidade são denominadas *classes derivadas*, pois estas últimas derivam (ou herdam) as propriedades da classe base.

A UML também define *diagramas de objetos*, que são similares aos diagramas de classes exceto pelo fato de que modelam objetos e suas relações. Em sistemas que criam e destroem objetos dinamicamente, diagramas de objetos são úteis para informar quais

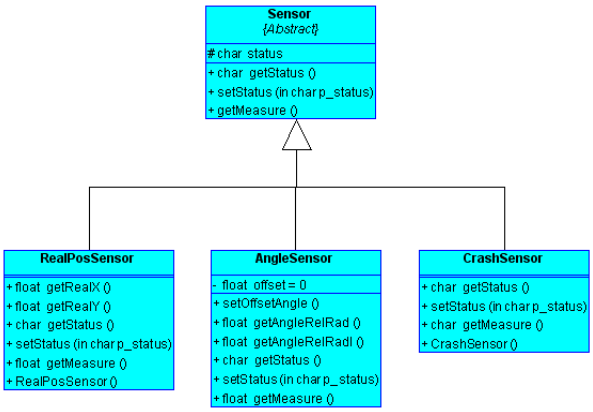

Figura 5.3: Exemplo de herança em diagrama de classes.

objetos estão participando do sistema em uma determinada situação ou instante de tempo. Os diagramas de objetos são iguais aos diagramas de classes, com a diferença de que os retângulos apresentam o nome das instâncias junto ao nome das classes.

A partir da definição das classes que compõem o sistema, elas começam a ser refinadas através da definição dos seus atributos e operações, e do modo como os objetos destas classes se comportam e interagem entre si, através da troca de mensagens.

5.2.3 Modelagem das Interações entre Objetos

Através das interações, objetos de classes diferentes se comunicam através da troca de mensagens, solicitando os serviços fornecidos pelas operações destes objetos. Interações entre objetos são modeladas em UML através de *diagramas de seqüência* e *diagramas de colaboração*. O diagrama de seqüência modela como as mensagens são enviadas entre objetos ao longo do tempo e pode ser utilizado para modelar a evolução temporal das atividades em um caso de uso completo. A Figura 5.4 apresenta um exemplo de diagrama de seqüência.

Em diagramas de seqüência, a linha tracejada que se estende para baixo, a partir do retângulo da classe ou objeto, é a *linha de vida* deste objeto e representa a progressão no tempo no sentido de cima para baixo. O envio de uma mensagem entre dois objetos é representado através de uma linha cheia com uma seta na extremidade do objeto que recebe a mensagem. O nome da mensagem aparece acima da linha da mensagem. A devolução do fluxo de controle ou o retorno de um valor é representado por uma linha

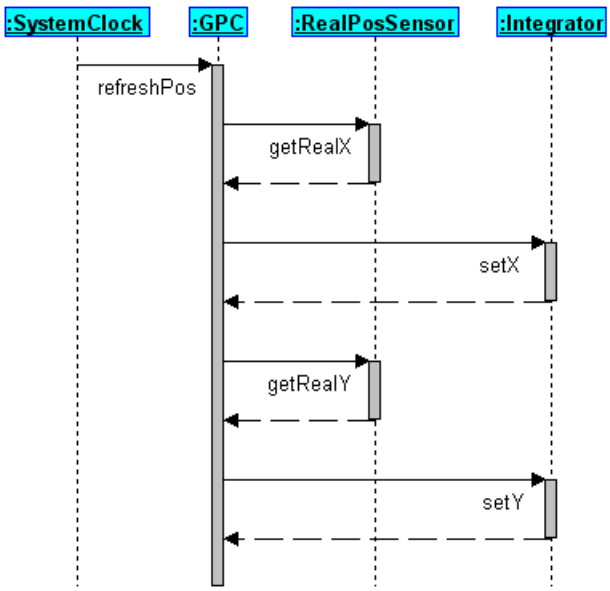

Figura 5.4: Exemplo de diagrama de seqüência.

tracejada com uma seta que parte do objeto que devolve o controle para o objeto que enviou a mensagem. Os retângulos verticais ao longo da linha de vida dos objetos são chamados de *ativações* e representam a duração de uma atividade. Uma ativação é iniciada a partir da chegada de uma mensagem e o comprimento vertical do retângulo corresponde à duração desta atividade. Um retângulo sobreposto ao outro indica a ativação de um método interno do próprio objeto.

A UML ainda oferece outra forma de modelar interações entre objetos, por meio de diagramas de colaboração. Estes diagramas também expressam a seqüência da troca de mensagens, como mostra o exemplo da Figura 5.5. Apesar destes diagramas não serem claros quanto à seqüência temporal da troca de mensagens, os mesmos são bastante úteis quando se deseja expressar fluxos de controle concorrênte em um mesmo cenário.

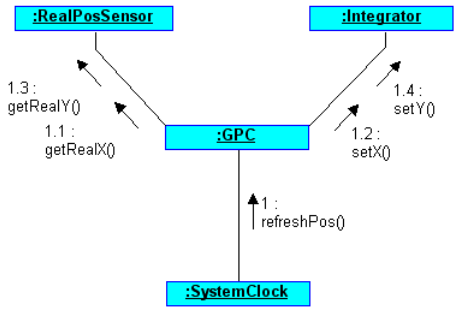

Figura 5.5: Exemplo de diagrama de colaboracao.

5.2.4 Modelagem do Comportamento Interno de Objetos

A modelagem do comportamento interno trata do comportamento dos objetos por uma ótica diferente da modelagem de interações. Enquanto a modelagem de interações aborda a comunicação entre diferentes objetos, a análise do comportamento interno trata do comportamento dos objetos de uma classe específica. A modelagem do comportamento interno analisa os diferentes eventos e as conseqüentes trocas de *estados* que podem ocorrer com objetos ao longo do tempo, em contraste à modelagem de interações que é definida a partir de casos de uso ou cenários específicos. Estados descrevem a condição de um objeto em um determinado instante de tempo.

Em UML, estados são modelados por meio de *diagramas de estados* ou *statecharts*, que permitem expressar como e sob quais condições os objetos de um sistema mudam de estado. A Figura 5.6 apresenta um exemplo de diagrama de estados.

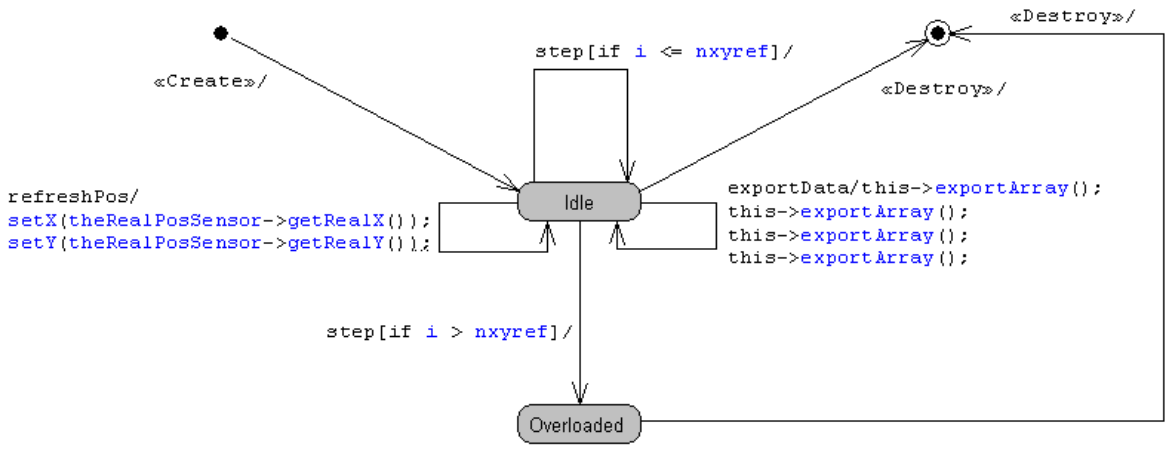

Figura 5.6: Exemplo de diagrama de estados.

Cada estado em um diagrama de estados é representado por um retângulo com cantos arredondados, com o nome do estado dentro. Um círculo cheio originando uma seta representa a condição inicial. Um círculo cheio circunscrito a outro apontado por uma seta indica o ponto final do diagrama. As linhas cheias com setas indicam *transições* que ocorrem em resposta a *eventos*.

Assim, a partir da modelagem da estrutura do sistema e do comportamento interno e interativo das classes e objetos, tem-se grande parte das informações necessárias para guiar a implementação do código da aplicação. Para completar a análise do sistema,

deve ser feita a análise de todas as restrições necessárias à implementação em tempo-real. A próxima seção apresenta as principais restrições presentes em aplicações de tempo-real, e que são utilizadas no escopo deste trabalho.

5.3 Restrições de Tempo-Real

Para garantir-se a previsibilidade e a confiabilidade temporal de uma aplicação, aspectos relativos a restrições temporais e mecanismo de garantia devem ser analisados. Comportamento temporal, relações de precedência entre tarefas, controle de acesso a recursos e estratégia de escalonamento são aspectos fundamentais que devem ser bem definidos de modo a garantir o correto funcionamento do sistema. Esta seção apresenta as principais restrições presentes em sistemas de tempo-real, com base no trabalho de Buttazzo (BUTTAZZO, 1997), e também como estas podem ser expressas em diagramas UML através de *esteriótipos* (stereotypes) definidos no perfil UML-TR.

5.3.1 Restrições Temporais

Tarefas computacionais de tempo-real são caracterizadas por apresentarem rigorosas restrições temporais que devem ser cumpridas de modo que o sistema controlado apresente o comportamento desejado. Uma restrição típica de uma tarefa é o *deadline*, que representa o tempo limite em que esta deve ter sua execução concluída sem causar nenhum dano ao sistema. Dependendo da conseqüência da perda de um *deadline*, as tarefas de tempo real podem ser classificadas como:

- *Hard:* Uma tarefa é do tipo *hard* se a perda de seu *deadline* pode causar conseqüências catastróficas ao sistema. Portanto, a tarefa deve ser garantida *a priori* em um cenário de pior caso;
- *Soft:* Uma tarefa é classificada como *soft* se a perda do seu *deadline* ocasiona perda de performance mas não causa riscos ao sistema.

Além da classificação pela conseqüência da perda de *deadline*, em *hard* ou *soft*[1], outros parâmetros são utilizados para caracterizar uma tarefa de tempo-real, como os que podem ser identificados na Figura 5.7. São eles:

- Tempo de chegada a_i: é o instante de tempo que a tarefa se torna apta para a execução. Também é conhecido como tempo de requisição ou tempo de lançamento, sendo denotado por r_i;
- Tempo de computação C_i: é o tempo necessário para o processador executar por completo a tarefa sem ser interrompido;
- Deadline d_i: é o tempo limite para a conclusão da tarefa de modo a não causar danos ou perda de performance ao sistema;
- Tempo de início s_i: é o tempo em que a tarefa tem a sua execução iniciada;
- Tempo de finalização f_i: é o tempo em que a tarefa tem a sua execução finalizada;
- Valor v_i: representa a importância relativa da tarefa em relação a outras tarefas do sistema;
- Atraso L_i: $L_i = f_i - d_i$ representa o atraso na finalização da execução da tarefa em relação ao seu *deadline*; pode-se observar que se uma tarefa é concluída antes do seu *deadline*, o valor do atraso é negativo.
- Tempo excedente E_i: $E_i = max(0, L_i)$ é o tempo que uma tarefa permanece ativa após o seu *deadline*;
- Tempo de relaxamento X_i: $X_i = d_i - a_i - C_i$ é o tempo máximo que a execução de uma tarefa pode ser atrasada sem que haja a perda do *deadline*;
- Regularidade: tipicamente, uma tarefa podem ser classificada quanto à sua regularidade como *periódica* (denotada por τ_i) e *aperiódica* (denotada por J_i).

O tempo de chegada da k*ésima* instância de uma tarefa periódica τ_i é dado por $a_i = \phi_i + (k-1)T_i$, no qual ϕ_i é a fase da tarefa ou instante da primeira ativação

[1]Alguns autores se referem à classificação em *hard* ou *soft* com índices de *criticalness*, vocábulo que não apresenta uma tradução direta para a língua portuguesa.

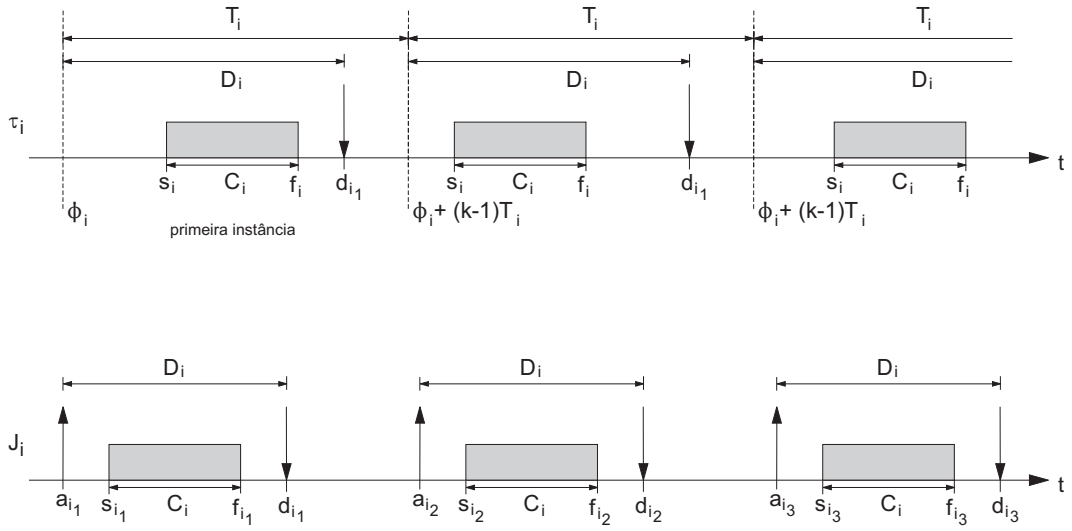

Figura 5.7: Parâmetros de tarefas de tempo-real.

e T_i é o período da tarefa. Na prática, uma tarefa periódica pode ser caracterizada pelo seu tempo de computação C_i e o seu *deadline* relativo D_i, que é muitas vezes considerado coincidente com o fim do período T_i. Além disso, os parâmetros C_i, T_i e D_i são considerados constantes a cada instância da tarefa (BUTTAZZO, 1997).

5.3.2 Relações de Precedência

Em certas aplicações, as atividades computacionais não podem ser executadas em qualquer ordem, e sim devem obedecer certas relações de precedência definidas na etapa de projeto do sistema. Por exemplo, em um sistema de controle clássico, a tarefa de *medição*, responsável por fazer a leitura das condições atuais das saídas do processo controlado, deve ser finalizada para que possa ser iniciada a execução da tarefa *cálculo da ação de controle*. Assim, o sistema deve ter mecanismos que permitam respeitar as relações de precedência e, quando necessário, ser capaz de lidar com variações nestas relações devido ao surgimento de tarefas não periódicas em momentos esporádicos.

Relações de precedência são normalmente representadas através de um grafo de precedência, onde tarefas são representadas por nós e relações de precedência por flechas.

5.3.3 Restrições de Recursos

Do ponto de vista de um processo, um *recurso* é qualquer estrutura de software que deve ser usada pelo processo para avançar sua execução (BUTTAZZO, 1997). Como exemplo de recursos típicos, podem ser citados estruturas de dados, conjunto de variáveis, áreas de memória, arquivos, trechos de código ou registradores de dispositivos periféricos. Um recurso dedicado a um processo particular é dito *recurso privado* enquanto que um recurso que pode ser utilizado por mais de uma tarefa é denominado *recurso compartilhado*.

Para manter a consistência de dados, muitos recursos compartilhados necessitam de exclusão mútua entre tarefas concorrentes que desejam acessá-lo. Estes são chamados de *recursos exclusivos*. Por exemplo, supõe-se que R é um recurso exclusivo compartilhado pelas tarefas J_a e J_b. Se A é uma operação realizada em R por J_a e B é uma operação realizada em R por J_b, então A e B nunca devem ser executadas ao mesmo tempo. O trecho de código executado em exclusão mútua é chamado de *seção crítica*. Assim, duas ou mais tarefas apresentam restrições de recursos quando estas compartilham recursos exclusivos.

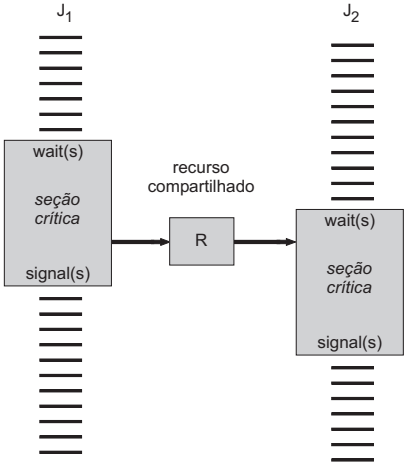

Figura 5.8: Estrutura de duas tarefas que compartilham o mesmo recurso.

Para assegurar o acesso seqüencial a recursos exclusivos, sistemas operacionais dispõem de mecanismos de sincronização como semáforos por exemplo, que podem ser utilizados nas seções críticas das tarefas. No caso de um semáforo binário s, a seção crítica de uma tarefa deve iniciar com a primitiva $wait(s)$ e terminar com a primitiva

signal(s), como mostra o exemplo da Figura 5.8. A primeira primitiva bloqueia um recurso para ser utilizado ao longo da seção crítica e a segunda libera o recurso. Quando uma tarefa em execução chama uma primitiva *wait* em semáforo bloqueado, ela entra no *estado de espera* até que a tarefa que bloqueou o recurso chame uma primitiva *signal*. Uma tarefa esperando por um recurso exclusivo é uma tarefa *bloqueada* neste recurso. É importante observar que um semáforo pode ocasionar uma situação em que uma tarefa de maior prioridade fique esperando a liberação de um recurso por uma tarefa de menor prioridade.

5.4 Algoritmo de Escalonamento

Um dos mecanismos fundamentais dos sistemas operacionais de tempo real (SOTR) é o escalonador, responsável por alocar recursos às tarefas do sistema de modo a completar a execução destas dentro das restrições impostas. Em sistemas operacionais convencionais, os algoritmos de escalonamento são desenvolvidos a partir do pressuposto de que as tarefas são atividades desconhecidas e ativadas de forma randômica. Exceto pela prioridade das tarefas, nenhum outro parâmetro é fornecido ao sistema, de modo que tempos de computação, restrições temporais, relações de precedência e compartilhamento de recursos não são levados em conta pelo escalonador. Desta forma, não há como garantir que as restrições temporais presentes em um sistema tempo-real sejam cumpridas.

Em um sistema de controle de tempo-real, o código de cada tarefa é conhecido e, portanto, pode ser analisado e testado, a fim de determinar suas características como tempo de computação, recursos utilizados, relações de precedência, entre outras. Assim, não há necessidade de se considerar tarefas como processos desconhecidos. Pelo contrário, os parâmetros conhecidos podem ser usados para definir as características mais apropriadas para mecanismos do SOTR, bem como o escalonador deste pode utilizar os parâmetros para verificar a factibilidade do escalonamento do conjunto de tarefas dentro das restrições temporais especificadas.

O grande número de algoritmos propostos para escalonar tarefas de tempo-real seguem a seguinte classificação:

- *Preemptivo*: em algoritmos preemptivos, a tarefa em execução pode ser interrompida a cada instante para atribuir o processador a outra tarefa ativa, de acordo com as regras de escalonamento.

- *Não-preemptivo*: em algoritmos não preemptivos, uma tarefa, uma vez iniciada, é executada pelo processador até o seu término. Neste caso, todas as decisões de escalonamento são tomadas ao término da execução das tarefas.

- *Estático*: algoritmos estáticos tomam as decisões de escalonamento com base em parâmetros fixos que são atribuídos antes da ativação das tarefas.

- *Dinâmico*: algoritmos dinâmicos tomam as decisões de escalonamento com base em parâmetros que podem mudar ao longo da operação.

- *Off-line*: em um algoritmo off-line, as decisões de escalonamento de um conjunto de tarefas são tomadas antes de qualquer ativação. O modelo de escalonamento gerado é armazenado em uma tabela para ser posteriormente executado por um *dispatcher*.

- *On-line*: um algoritmo on-line toma as decisões de escalonamento em tempo de execução a cada chegada de uma nova tarefa ou sempre que uma tarefa termina a sua execução.

- *Ótimo*: um escalonamento é dito ótimo quando toma suas decisões de modo a minimizar uma função custo definida para um conjunto de tarefas.

- *Heurístico*: um algoritmo é dito heurístico quando tende a garantir o escalonamento ótimo, embora garantias temporais não possam ser asseguradas.

Em aplicações de tempo-real com características dinâmicas, a garantia da execução das tarefas deve ser feita em tempo de execução, a cada chegada ou conclusão de uma tarefa. A cada chegada de uma tarefa, a decisão de aceitá-la ou rejeitá-la deve ser tomada considerando-se o cenário de pior caso. Por exemplo, uma tarefa recém chegada só pode ter sua execução iniciada em um período de amostragem, se mesmo nas piores condições haverá a garantia de finalização das demais tarefas até os seus respectivos *deadlines*. Algoritmos que tomam decisões com base no pior caso são chamados de

algoritmos baseados em garantia ou *guarantee-based algorithms*. Embora este mecanismo de garantia certifique que, uma vez aceita, a tarefa será concluída dentro do seu *deadline* e não impedirá a conclusão das demais, é importante observar que, como as decisões são tomadas com base no cenário de pior caso, tarefas podem ser rejeitadas desnecessariamente.

Outra abordagem para o tratamento das tarefas em aplicações de tempo real são os *algoritmos de computação imprecisa*. A computação imprecisa procura tratar situações dinâmicas em que não haja tempo suficiente para finalizar a execução de todas as tarefas dentro dos seus respectivos *deadlines*. Nestes casos se procura utilizar os recursos disponíveis para produzir um resultado aproximado que possa ser utilizado de modo a evitar situações de risco. Em sistemas de TR que suportam computação imprecisa, toda a tarefa J_i é decomposta em uma subtarefa *mandatória* M_i e uma tarefa *opcional* O_i. A subtarefa mandatória consiste na porção de J_i que deve ser computada de modo a produzir um resultado de qualidade aceitável, enquanto que a subtarefa opcional refina o resultado. Tanto M_i quanto O_i tem o mesmo tempo de chegada a_i e o mesmo *deadline* d_i da tarefa original J_i, mas O_i se torna apta à execução apenas ao final da execução de M_i. Se C_i é o tempo de execução de J_i, M_i e O_i tem respectivamente tempos de execução m_i e o_i, de modo que $C_i = m_i + o_i$. Assim, de modo a garantir um nível mínimo de performance, M_i deve ser concluída até o seu *deadline* enquanto que O_i pode ficar incompleta se necessário, ao custo de uma perda de qualidade do resultado final.

É importante observar que o modelo de tarefas utilizado em sistemas tradicionais de tempo-real pode ser considerado um caso particular do modelo adotado em computação imprecisa. De fato, tarefas do tipo hard são aquelas que não apresentam parte opcional O_i e tarefas do tipo soft não apresentam parte mandatória M_i.

A próxima seção apresenta o Perfil para UML Tempo-Real e como ele permite expressar, em diagramas UML, as restrições de tempo-real apresentadas nesta seção.

5.5 Perfil para o UML Tempo-Real

O Perfil UML para Escalonabilidade, Performance e Tempo (UML Profile for Schedulability, Performance and Time) (OMG, 2002), foi desenvolvido pelo grupo de especialistas do OMG para análise e projeto de sistemas tempo-real. O principal requisito desta especificação é permitir a modelagem de sistemas determinísticos e previsíveis temporalmente com o uso da UML. Um sistema é caracterizado como determinístico quando as reações aos eventos podem ser quantificadas e conhecidas com antecedência. O mesmo é definido como previsível temporalmente quando suas características temporais são conhecidas e limitadas (BECKER, 2003). Parâmetros de tarefas de tempo-real como *deadlines*, tempo de computação entre outros ilustrados na Figura 5.7, são exemplos destas características. Assim, o Perfil para UML tempo-real (UML-TR) torna a linguagem UML expressiva em relação a informações relacionadas ao tempo e, desta forma, adequada a modelagem de sistemas tempo-real.

O perfil UML-TR é composto por uma estrutura hierárquica contendo uma parte genérica, comum a todos os elementos do modelo, e também partes especializadas, que abordam requisitos e funcionalidades específicas. A parte genérica é denominada *Modelo Geral de Recursos*, cujo núcleo é um *framework* para a modelagem de recursos. Dois outros *frameworks* se derivam do primeiro, um para modelagem de tempo e outro para modelagem de concorrência. As subpartes especializadas são divididas em um módulo para definição de métodos de análise e outro para definição da infra-estrutura de execução. O módulo contendo os métodos de análise sugere um *framework* para análise de escalonabilidade, enquanto o segundo sugere o mapeamento para um ambiente de execução específico. Maiores detalhes sobre a estrutura hierárquica do perfil UML-TR e suas partes são apresentadas em BECKER (2003). Para expressar os requisitos de tempo-real descritos na Seção 5.3, são utilizados os *frameworks* para modelagem de tempo e concorrência, que serão apresentados a seguir.

5.5.1 Framework para Modelagem de Tempo

O *framework* para modelagem de recursos introduz conceitos para a especificação de restrições temporais, tais como definição do tempo máximo de resposta de um método

ou atividade. Estes conceitos são associados a estereótipos iniciados pelo prefixo *RT*.

Este framework é particionado em quatro módulos distintos, porém relacionados entre si. São eles:

1. Módulo para modelagem de tempo e valores de tempo (TimeModel);
2. Módulo para modelagem de mecanismos de tempo (TimingMechanisms);
3. Módulo para modelagem de eventos no tempo (TimedEvents);
4. Módulo para modelagem de serviços temporais (TimingServices).

O módulo para modelagem de tempo discute os detalhes relacionados com a conceituação de tempo. É definido neste módulo o conceito de valores de tempo (*time value*), que se referem a uma medida de tempo. Também é definido o conceito de duração (*duration*), que representa o tempo passado entre dois instantes, sendo representado neste módulo por um intervalo de tempo (*time interval*). O módulo para modelagem dos mecanismos de tempo oferece *timers* e *clocks* para a modelagem da infraestrutura e mecanismos temporais. Os *timers* podem ser programados para gerarem um evento quando um determinado ponto no tempo for alcançado. O instante de tempo para geração do evento pode ser tanto definido de forma absoluta (tal como, "às 10 horas") ou relativa ("após 15 minutos"). Já os clocks são mecanismos que geram periodicamente uma interrupção. Os eventos, definidos no módulo para modelagem de eventos no tempo, são caracterizados por não possuírem duração, sendo associados apenas com um instante no tempo. Um tipo especial de evento é denominado *timed event*, que indica a chegada em algum instante de tempo pré-determinado. Isto pode ser conseqüência da expiração de um certo intervalo de tempo, como *timeout*, ou da leitura de um clock. Por fim, o modelo para modelagem de serviços temporais trata daqueles mecanismos essenciais para a programação de requisitos temporais, geralmente presentes nos sistemas operacionais tempo-real. Como exemplos destes serviços, pode-se citar as operações de leitura e acerto de relógios e também as operações de criação e manutenção de *timers*.

5.5.2 Framework para Modelagem de Concorrência

Este *framework* permite a modelagem de aspectos referentes à execução concorrente, uma característica presente na maioria dos sistemas tempo-real, visto que estes apresentam uma forte interação com o meio físico, que é inerentemente concorrente. Os estereótipos definidos neste framework iniciam com o prefixo *CR*. O mesmo se encontra dividido nos seguintes módulos:

1. Recursos concorrentes: são os recursos que representam os mecanismos para um comportamento concorrente (ou pseudo-concorrente) no sistema, sendo representados no sistema operacional por processos ou tarefas;

2. Cenários concorrentes: representam seqüências de ações geralmente interligadas efetuadas por recursos concorrentes;

3. Serviços de recursos concorrentes: representam serviços que possuem algum tipo de política de controle de acesso que os protegem contra os efeitos indesejados da própria concorrência.

Recursos concorrentes (representado por *ConcurrentUnit* no modelo conceitual) são modelados como recursos ativos que, a partir da criação, iniciam a executar um cenário principal denominado *main* e continuam até o cenário terminar ou até que seja explicitamente abortado. Este elemento pode possuir uma ou mais filas para armazenar mensagens ou estímulos vindos de outros recursos, denominadas *StimuliQueue*, as quais não necessitam ser imediatamente processadas, i.e. são deferidas (serviçoes denominados *deferred*). Por outro lado, serviços *immediate* implicam a criação de um novo cenário de execução, o qual necessita de uma unidade concorrente auxiliar para poder executar. Esta unidade pode ser criada por demanda ou pré-alocada em filas (*pools*) dela mesma.

O modelo de concorrência faz ainda a distinção entre as ações de comunicação, ou seja, solicitações de serviços que podem ser síncronas ou assíncronas. A primeira bloqueia o cliente até que o serviço invocado seja completado, enquanto na segunda o cliente permanece com o controle do fluxo de execução.

5.5.3 Framework para Modelagem de Escalonabilidade

Este *framework* é derivado do *framework* para modelagem de concorrência e oferece suporte à análise de escalonabilidade do modelo, permitindo determinar se as características de qualidade de serviço (QoS) solicitadas pelos clientes podem ser atendidas pelos recursos. Os estereótipos definidos neste *framework* iniciam com o prefixo *SA* e permitem análisar questões referentes a ambientes de execução, recursos protegidos, estímulos e respostas, além de caracterísitcas de escalonamento relevantes como pior caso de tempo de execução (*Worst Case Execution Time* - WCET) entre outras.

5.5.4 Aplicação a Diagramas UML

A Figura 5.9 apresenta exemplos de como requisitos e parâmetros de tempo-real podem ser representados em diagramas UML por meio do uso dos esteriótipos do perfil UML-TR.

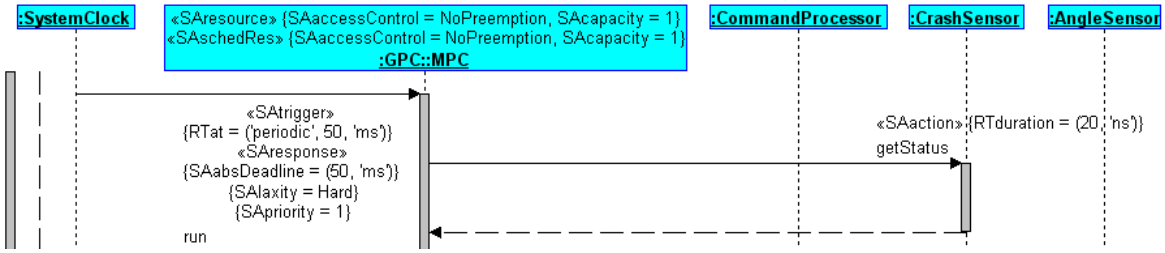

Figura 5.9: Exemplo de uso do perfil UML-TR em diagrama UML.

Como mostra a figura, o esteriótipo *SAtrigger* representa um gatilho que dispara a execução de uma tarefa em uma unidade de escalonamento, oferecendo marcas para definição do padrão de ativação. No exemplo, a marca *RTat* indica um estímulo periódico com ativação a cada $50ms$.

A execução da tarefa caracteriza uma resposta, cujas propriedades são definidas através das marcas oferecidas pelo estereótipo *SAresponse*. Na Figura 5.9, o tipo e o valor do *deadline* são definidos respectivamente pelas marcas *SAlaxity* e *SAabsDeadline*. Também é usada a marca *SApriority* para definir a prioridade de execução. Estas marcas juntamente com outras oferecidas pelo esterótipo *SAresponde*, permitem que sejam representados todos os parâmetros de tarefas de tempo-real apresentadas na Seção 5.3.

Parâmetros específicos das ações que compõem a execução da resposta são representadas pelo estereótipo *SAaction*, que no diagrama da Figura 5.9 é empregado para definir o tempo de duração das ações através da marca *RTduration*.

As unidades de escalonamento, responsáveis pela execução das tarefas, são representadas por classes decoradas com o estereótipo *SAschedRes*, conforme mostra a Figura 5.9. Também o exemplo desta figura mostra a utilização do estereótipo *SAresource*, que representa um recurso compartilhado cujo acesso deve ser exclusivo e sem preempção, conforme definem as marcas *SAcapacity* e *SAaccessControl*.

5.6 Conclusões

Este capítulo apresentou os principais conceitos empregados na computação em tempo-real orientada a objetos. Foi apresentada a UML como linguagem padrão para modelagem de sistemas orientados a objetos, bem como as restrições típicas presentes em sistemas de tempo-real e como estas podem ser expressas na UML através do perfil UML-TR.

A UML, linguagem gráfica padronizada pelo OMG para modelagem de sistemas OO, foi definida como ferramenta a ser utilizada para modelagem do sistema de controle de seguimento de trajetória, sendo apresentados os diagramas oferecidos para a modelagem de aspectos do sistema, tais como estrutura, interação e comportamento dinâmico das entidades (objetos) que o compõem. Com relação às características de tempo-real, aspectos como restrições temporais, relações de precedência e restrições de recursos foram apresentados. Por fim, foram apresentados os conceitos básicos do Perfil UML Tempo-Real, que permite expressar de forma padronizada, restrições de tempo-real em diagramas UML.

No próximo capítulo é apresentada a modelagem orientada a objetos do sistema de controle de seguimento de trajetória, utilizando os conceitos aqui descritos, de modo a orientar a implementação de código aplicável às plataformas de sistemas embarcados.

Capítulo 6

Modelagem e Descrição do Problema

6.1 Introdução

Este capítulo tem o objetivo de apresentar a análise e projeto do sistema de controle para seguimento de trajetória de robôs móveis, a partir do estudo realizado no Capítulo 4. A análise e o projeto foram feitos com base no paradigma de orientação a objetos direcionado a aplicações de tempo-real, cujos conceitos foram apresentados no Capítulo 5. Através dos princípios da modelagem OO, busca-se chegar a um sistema modular, de fácil adaptação e manutenção, e que permita a reutilização das classes em outras aplicações de controle de processos. Devido às características de tempo-real do problema, foram também modeladas as restrições temporais necessárias à aplicação, como *deadlines*, exceções e prioridades das tarefas. Como a implementação do sistema pode ter como destino diferentes plataformas embarcadas, a modelagem também leva em consideração questões de portabilidade, de forma a facilitar implementações em diferentes plataformas, como processadores digitais de sinais (DSPs), microcontroladores e microprocessadores em geral.

Inicialmente, é apresentada a descrição detalhada do problema, onde são identificadas as funcionalidades que são requisitos para sistemas de controle de trajetória para robôs, além das atividades auxiliares que agregam serviços úteis à aplicação. A partir da definição das funcionalidades, é apresentada a identificação e modelagem das restrições temporais necessárias para garantir a previsibilidade do sistema quanto à execução correta das tarefas computacionais. Em seguida, é apresentado o modelo OO

UML, desenvolvido com base nas funcionalidades e restrições do sistema. Por fim, são descritas as características necessárias ao mecanismos de escalonamento do sistema.

6.2 Levantamento dos Requisitos

A primeira etapa de um projeto de software consiste na descrição detalhada dos requisitos do problema. Esta prática assegura que as características do sistema estejam bem definidas para as fases de projeto e desenvolvimento, diminuindo a chance de haver necessidade de mudanças no projeto durante a implementação, o que normalmente implica custos e desperdício de tempo (DEITEL e DEITEL, 2001).

O problema aqui tratado consiste em um sistema de controle para seguimento de trajetória de veículos direcionais e diferenciais, com base na técnica de controle preditivo utilizando sistemas de coordenadas locais, apresentada no Capítulo 4. A partir desta definição, são listadas funcionalidades necessárias ao sistema.

6.2.1 Funcionalidades do Sistema de Controle

- **Controle de Trajetória**

  Funcionalidade baseada em algoritmo de controle preditivo para controlar a trajetória de deslocamento de veículos direcionais e diferenciais, seguindo uma trajetória de referência. Necessita de medição da velocidade de deslocamento linear do veículo, sistema de posicionamento das rodas (veículo direcional) ou sistema de controle diferencial para as rodas (veículo diferencial), medição do ângulo de orientação do veículo, sistema de geração de trajetória de aproximação e integrador das coordenadas cartesianas do deslocamento;

- **Controle de Velocidade**

  Funcionalidade responsável por manter a velocidade do veículo constante, de acordo com referências pré-determinadas, em diferentes condições de carga e inclinação de pista. Necessita de medição de velocidade de deslocamento do veículo. Neste trabalho, o sistema de controle de velocidade é considerado um sistema independente que opera em conjunto com o Controle de Seguimento de Trajetória;

- **Sistema Anti-Colisão**

  Funcionalidade que atua em situações de eminente colisão ou crescimento do erro de trajetória além de limites de segurança, freando imediatamente o veículo.

- **Sistema de Contorno de Obstáculos**

  Através de algoritmo específico, este sistema deve ser capaz de gerar trajetórias de referência alternativas a trajetória original, de forma a contornar eventuais obstáculos, ou gerar trajetórias de aproximação para casos em que o veículo esteja fora da trajetória desejada. É importante observar que sistemas de contorno de obstáculos adicionam tarefas aperiódicas ao sistema de controle, de modo que o escalonador deve garantir o cumprimento de todas as restrições de tempo-real. O escopo deste trabalho não abrange sistemas de contorno de obstáculos;

- **Sistema de Posicionamento Real**

  Tem a finalidade de informar periodicamente as coordenadas da posição real do veículo para corrigir o erro de integração do sistema de controle. Pode fazer uso de sistemas de posicionamento global (GPS) ou sistemas de apoio na pista, como *cabo-guia* entre outros;

- **Seleção de Trajetórias**

  Banco de dados com trajetórias pré-determinadas e métodos para geração de trajetórias com base em parâmetros definidos pelo usuário. Permite que o operador programe trajetórias de deslocamento, e as armazene em base de dados local, bem como obtenha trajetórias de bancos de dados remotos, via sistema de comunicação. A referência é composta pelos valores das coordenadas cartesianas x e y e do ângulo de orientação θ para cada ponto ao longo da trajetória;

- **Ajuste de Modos de Operação**

  Os modos de operação são caracterizados por diferentes configurações do sistema de controle que podem ser selecionadas pelo operador, de modo a realizar uma sintonia específica de parâmetros, como horizonte de predição e controle, ponderação do erro futuro e do esforço de controle e ajuste da distância *look-ahead*, além da definição do modelo do processo, configurando o processo de condução para a atender requisitos como rapidez, economia e segurança entre outros. As-

sim, combinações específicas de sintonia destes parâmetros podem dar origem a modos de operação específicos, como por exemplo:

- Modo de operação normal: Consiste no modo de operação que basicamente procura um melhor compromisso entre rapidez, economia e segurança;
- Modo de operação econômico: As configurações do sistema de controle podem procurar manter o veículo sob condições que proporcionem o maior rendimento possível, minimizando o consumo de energia e aumentando a vida útil dos sistemas eletro-mecânicos. Partes do sistema que sejam dispensáveis para os controladores podem ser desligas em situações críticas para poupar energia;
- Modo de operação otimizado: Nesta configuração, as ações de controle tem maior liberdade de forma a tornar o deslocamento do veículo mais ágil.

A seleção do modo de operação pode ser feita off-line ou on-line dependendo das necessidades da aplicação. É importante observar que o ajuste do modo de operação pode ter influência sobre o tempo de execução das tarefas do sistema, implicando a necessidade de cuidado na determinação dos parâmetros a fim de preservar a viabilidade de implementação em tempo-real.

- **Sistema de Comunicação**
  Comunicação baseada em transmissor RF, WAN; Viabilizaria a implantação de sistemas de controle de tráfego, deslocamento em comboio, etc. O escopo deste trabalho não abrange o detalhamento e a implementação de sistemas de comunicação;

- **Interface com o Operador**
  - Seleção de modos de operação;
  - Seleção de velocidade;
  - Seleção de trajetória via banco de dados local ou remoto via sistema de comunicação;
  - Informação de velocidade;

- Informação de posição;
- Informação de temperatura de operação (temperatura do motor, etc);
- Identificação de níveis de bateria/combustível.

Dentre as funcionalidades descritas acima, algumas são indispensáveis para o funcionamento do sistema enquanto que outras são apenas auxiliares. Os controles de trajetória e velocidade, o sistema anti-colisão, o ajuste de modo de operação e a seleção de trajetória são funcionalidades indispensáveis. Dentre estas, as três primeiras devem necessariamente ser executadas em tempo-real, enquanto que as duas últimas podem ser executadas off-line em aplicações cujas condições de operação, como velocidade e trajetória, permanecem constantes. O sistema de posicionamento real é necessário em aplicações de alta precisão, nas quais os níveis de erro de integração não são aceitáveis. O sistema de contorno de obstáculos é util em aplicações em ambiente dinâmico, onde a posição dos obstáculos não pode ser pré-determinada. Também o sistema de comunicação e a interface com o operador são dispensáveis em algumas aplicações.

Neste trabalho foram implementadas as funcionalidades indispensáveis para o funcionamento do controle de seguimento de trajetória, e as operações necessárias à comunicação com alguns sistemas auxiliares, sendo eles o sistema de posicionamento real e a interface com o operador. Sistemas de controle de velocidade, como já mencionado, são considerados sistemas independentes, a exemplo de GOMES (2003).

6.3 Restrições de Tempo-Real

A partir da definição das funcionalidades necessárias ao sistema, é realizada a identificação das restrições temporais da aplicação, necessárias para garantir a previsibilidade do sistema. Desta forma, na modelagem das tarefas do sistema foram analisadas questões referentes às restrições temporais, relações de precedência, restrições de recursos e tratamento de exceções. Estas questões direcionam o estudo para a análise da necessidade de sistemas operacionais de tempo-real, ou SOTR, sendo discutidas características de alguns mecanismos importantes como o algoritmo de escalonamento e o tratador de interrupções.

Com base nas funcionalidades do sistema, foram identificadas as principais tarefas computacionais necessárias ao SCST. Estas tarefas são:

- τ_1 - Cálculo da ação de controle para o modelo cinemático;
- J_2 - Definição do modo de operação;
- J_3 - Seleção da trajetória;
- τ_4 - Correção do erro de integração.
- J_5 - Envio de informações;
- J_6 - Início do percurso da trajetória;
- J_7 - Interrupção do percurso da trajetória;
- τ_8 - Cálculo da ação de controle para a dinâmica de β e $\dot{\theta}$;
- τ_9 - Cálculo da ação de controle para a dinâmica da velocidade no centro de massa;

A tarefa J_2 realiza as operações que constituem a funcionalidade *Ajuste de Modo de Operação*, J_3 consiste nas operações que implementam a funcionalidade *Seleção de Trajetórias*, enquanto que J_5, J_6 e J_7 realizam outras operações oferecidas pela funcionalidade *Interface com o Operador*. A funcionalidade *Sistema de Posicionamento Real*, quando necessária, é implementada pela tarefa τ_4.

Em aplicações baseadas em modelo cinemático, a tarefa τ_1 implementa as funcionalidades *Controle de Trajetória* e *Sistema Anti-Colisão*, e a funcionalidade *Controle de Velocidade* é implementada em sistema independente. Em aplicações de alto desempenho, as funcionalidades *Controle de Trajetória* e *Sistema Anti-Colisão* são implementadas pelas tarefas τ_1 e τ_8, e τ_9 implementa a funcionalidade *Controle de Velocidade*.

Dentre estas tarefas, τ_1 deve necessariamente ser executada a cada período de amostragem para guiar o veículo ao longo da trajetória. O mesmo vale para τ_8 e τ_9 em se tratando de aplicações de alto desempenho. As tarefas J_2 e J_3 podem ser executadas off-line caso seja utilizado uma plataforma com hardware mais restrito, de forma a reduzir o custo computacional e o volume de memória utilizado. Porém, sua execução

on-line proporciona mais flexibilidade e capacidade de adaptação do sistema frente a mudanças das condições de operação, pois, permite, em tempo de execução, alterações na configuração do sistema, na trajetória e no controlador. As tarefas τ_4 e J_5 não são estritamente necessárias em todas as aplicações. Já J_6 e J_7 comandam, respectivamente, o início e o fim do percurso.

Com isso, são identificados dois modos de implementação do sistema, denominados SCST Completo e SCST Simplificado. O primeiro modo consiste em uma implementação capaz de executar todas as tarefas do sistema. Já o segundo modo consiste em uma aplicação mais restrita, onde apenas as tarefas τ_1, J_6, J_7, τ_4 τ_8 e τ_9 são executadas. Este segundo modo é destinado a aplicações mais restritas onde a trajetória é fixa e as condições de operação sofrem poucas variações, permitindo que as tarefas J_2 e J_3 sejam executadas off-line e a configuração do sistema seja realizada em tempo de compilação. No modo simplificado, a execução off-line de J_2 e J_3 reduz o custo computacional e o uso de memória, possibilitando a implementação em plataformas com hardware mais simples. Em relação às tarefas τ_4, τ_8 e τ_9, observa-se que estas podem ou não estar presentes em ambos os modos, dependendo do nível de precisão necessário em cada aplicação.

A partir da definição das tarefas do sistema, foram identificadas as restrições temporais do sistema, tomando como base os aspectos discutidos no Capítulo 5.

6.3.1 Restrições Temporais

No contexto do SCST, as tarefas τ_1, τ_8 e τ_9 são do tipo *hard*, pois devem ter as suas execuções completadas a cada período de amostragem de modo a se obter o valor das ações de controle que conduzem o veículo. Uma perda de *deadline* destas tarefas pode fazer com que o veículo saia da trajetória, de modo que elas devem ter os maiores níveis de prioridade nas decisões de escalonamento. Assim, em uma aplicação baseada em modelo cinemático, τ_1 deve ter o nível máximo de prioridade, aqui definido como 1. Em aplicações de alto desempenho, τ_8 e τ_9 são definidas com prioridade 1 e τ_1 definida com prioridade 2. A opção por definir as tarefas de controle da dinâmica com nível máximo de prioridade e as tarefas do controle da cinemática com nível imediatamente inferior foi motivada pelo fato de que o período de amostragem da malha de controle da

dinâmica deve ser menor que o período de amostragem da malha da cinemática, o que implica que τ_8 e τ_9 possuem *deadlines* mais curtos. Também J_6 e J_7 são definidas como do tipo *hard* e com prioridade 1, de modo que os comandos de início e interrupção do percurso sejam atendidos imediatamente.

As demais tarefas, quando utilizadas, são do tipo *soft*. É importante mencionar que a tarefa J_2 é considerada do tipo *soft* pressupondo que, quando necessária, esta visa a melhoria do desempenho frente a mudanças nas condições de operação, e nunca visa a correção de erros significativos no modelo do processo de modo a contornar situações críticas. Da mesma maneira, pressupõe-se que o erro de integração não deve chegar a valores significativos a ponto de provocar uma situação crítica, devendo, quando necessário, ser corrigido ainda quando pequeno. Com a exceção de J_2, a prioridade das tarefas do tipo *soft* é definida como nível 3. Já J_2 tem nível de prioridade 4 de modo a não atrasar a execução das demais tarefas do tipo *soft*, pois seu tempo de execução pode ser muito elevado, como será detalhado mais adiante.

Uma prática importante para tratar restrições temporais em sistemas de tempo-real é a utilização de exceções, ou seja, trechos de código que são executados no caso da perda do *deadline* das tarefas, de modo a evitar situações de risco, ou pelo menos minimizar as conseqüências destas. No caso das tarefas do SCST, as tarefas τ_1 e τ_8 permitem a implementação de tratamento de exceções para a ação correspondente a leitura do ângulo de orientação θ. No caso de uma falha na comunicação com a bússola eletrônica, o valor atual de θ pode ser estimado através da equação de θ da expressão (4.4) para robô diferencial, ou da expressão (4.8) para robô direcional. Assim, esta possibilidade de tratamento de exceção deve ser considerada no projeto do algoritmo em tempo-real.

Quanto a regularidade, tem-se que as tarefas τ_1, τ_8 e τ_9 são periódicas, como a própria notação usada para identificá-las já dá a entender. Já a tarefa J_2 pode ser tanto periódica quanto aperiódica, dependendo das necessidades da aplicação. Neste trabalho, como se pode perceber pela notação utilizada, J_2 é considerada aperiódica pois não há a necessidade de se atualizar o modo de operação a cada período de amostragem, e sim em momentos esporádicos dependendo da necessidade. Porém, em aplicações de natureza adaptativa, nas quais o modelo do veículo é identificado em tempo de

execução, J_2 passam a ser tarefa essencialmente periódica. Da mesma maneira, as tarefas τ_4 e J_5, quando necessárias, podem ser tanto periódicas quanto aperiódicas, dependendo das necessidades específicas da aplicação. Além disso, quando periódicas, estas tarefas não necessariamente necessitam de um período de ativação igual ao do loop de controle, podendo ser ativadas a uma freqüência menor. Como se pode observar nas figuras 6.17 e 6.18 e também na notação utilizada, as tarefas τ_4 e J_5 foram modeladas como periódica e aperiódica respectivamente, sendo que J_5 é executada apenas com o robô em repouso. Por fim, J_3, J_6 e J_7 são neste projeto consideradas aperiódicas. Sobre J_3 tem-se que, assim como J_5, esta também é executada apenas com o robô em repouso.

Com relação aos tempos de computação das tarefas, estes são dependentes da plataforma de destino, sendo analisados no próximo capítulo.

6.3.2 Relações de Precedência

Com base na análise realizada na seção anterior, pode-se perceber que de modo geral não há relações de precedência entre as tarefas do SCST, exceto pelo fato natural de que J_7 só pode ser executada após J_6, ou seja, a interrupção do percurso só pode ser realizada após este já ter sido iniciado. Porém, em aplicações do SCST Completo, devem ser respeitadas algumas restrições em duas situações específicas.

A primeira delas é caracterizada pelo fato de que o percurso só pode ser iniciado após a definição do modo de operação e da trajetória. Isso implica que, embora durante a realização do percurso as tarefas τ_1, J_2, τ_4, J_7, τ_8 e τ_9 não possuam antecessoras e possam ter suas execuções iniciadas a qualquer instante de maneira concorrente entre si, a primeira instância de J_6 só pode ser executada após a execução de pelo menos uma instância das tarefas J_2 e J_3.

A segunda restrição está no fato de que as tarefas J_3, J_5 e J_6 só podem ser executadas com o robô em repouso e além disso, não podem ser concorrentes entre si, uma vez que não existe esta necessidade. Porém, J_3, J_5 e J_6 (esta a partir da sua segunda instância) não apresentam antecessoras e podem ser executadas de modo seqüencial em qualquer ordem.

Para o caso do SCST Simplificado, ocorre apenas que as tarefas J_3 e J_5, se utilizadas, devem assim como no modo completo, ser executadas apenas com o robô em repouso.

6.3.3 Restrições de Recursos

No SCST Completo, as áreas de memória que armazenam a definição do sistema, a lei de controle e a integração das posições do robô, são recursos compartilhados entre diferentes tarefas concorrentes. Estas áreas de memória são recursos exclusivos, pois devem ser protegidas por mecanismos que garantam exclusão mútua para evitar que operações de escrita e leitura ocorram de maneira simultânea ou sejam interrompidas antes do seu término.

As áreas de memória que contêm a definição do sistema e da lei de controle são acessadas pelas tarefas τ_1, J_2, τ_8 e τ_9, sendo que J_2 realiza a escrita e as demais realizam a leitura desta memória. Assim, um controle de acesso sem preempção é necessário para evitar que, por ter a menor prioridade, J_2 seja interrompida durante suas seções críticas por outras tarefas de maior prioridade. Isso acabaria por deixar dados inconsistentes na definição do sistema ou na lei de controle, dados estes que seriam perigosamente utilizados no cálculo da ação de controle.

A Figura 6.1 ilustra o acesso das tarefas do sistema aos recursos exclusivos, para o caso do SCST Completo baseado em modelo cinemático. No diagrama, paralelogramos representam as tarefas, retângulos com cantos arredondados representam os dispositivos de hardware e os desenhos estilizadas de piscinas representam os recursos compartilhados.

No caso específico do SCST em cascata para aplicações de alto desempenho, a área de memória utilizada pela instância da classe *MPC* para armazenar o horizonte de controle do modelo cinemático também caracteriza um recurso exclusivo, pois além do acesso de escrita pela própria instância de *MPC* (na execução de τ_1), esta área de memória também é lida, como referência, por uma instância da classe *BetaThetaMPC* (durante a execução de τ_8). Assim, de modo a evitar acessos simultâneos de leitura e escrita do horizonte de controle cinemático, novamente é necessário mecanismo de exclusão mútua.

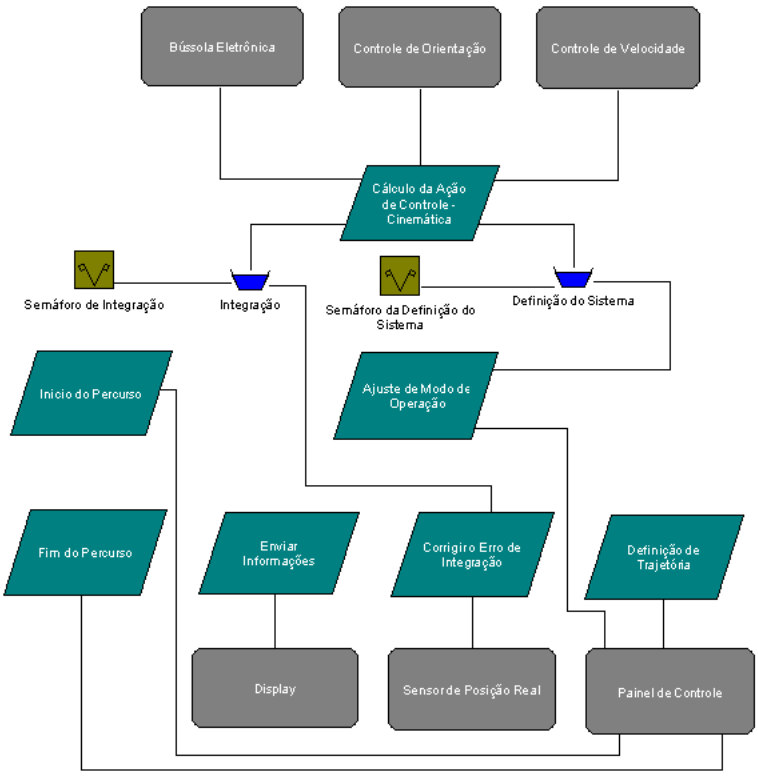

Figura 6.1: Recursos exclusivos utilizados pelas tarefas do SCST Completo baseado em modelo cinemático.

O mesmo mecanismo de controle de acesso ainda é necessário para a memória da integração das coordenadas do robô, em aplicações do SCST Completo ou Simplificado nas quais a correção do erro de integração seja necessária. Nestas aplicações, a área de memória da integração é acessada pelas tarefas τ_1 e τ_4, que realizam leitura e escrita respectivamente. A interrupção de τ_4 (que tem menor prioridade) por τ_1 (que tem prioridade máxima) pode fazer com que τ_1 utilize dados inconsistentes caso a seção crítica de τ_4 não tenha sido completada.

A Figura 6.2 ilustra o acesso das tarefas do sistema aos recursos exclusivos, para o caso do SCST Completo com estrutura em cascata. A notação utilizada é a mesma da Figura 6.1.

Em todos estes casos é importante observar que, quando necessário, um mecanismo de controle de acesso a recursos exclusivos que não permita a preempção durante a execução da seção crítica das tarefas pode ocasionar uma situação em que uma tarefa de maior prioridade tenha que esperar pela conclusão de uma tarefa de menor prioridade.

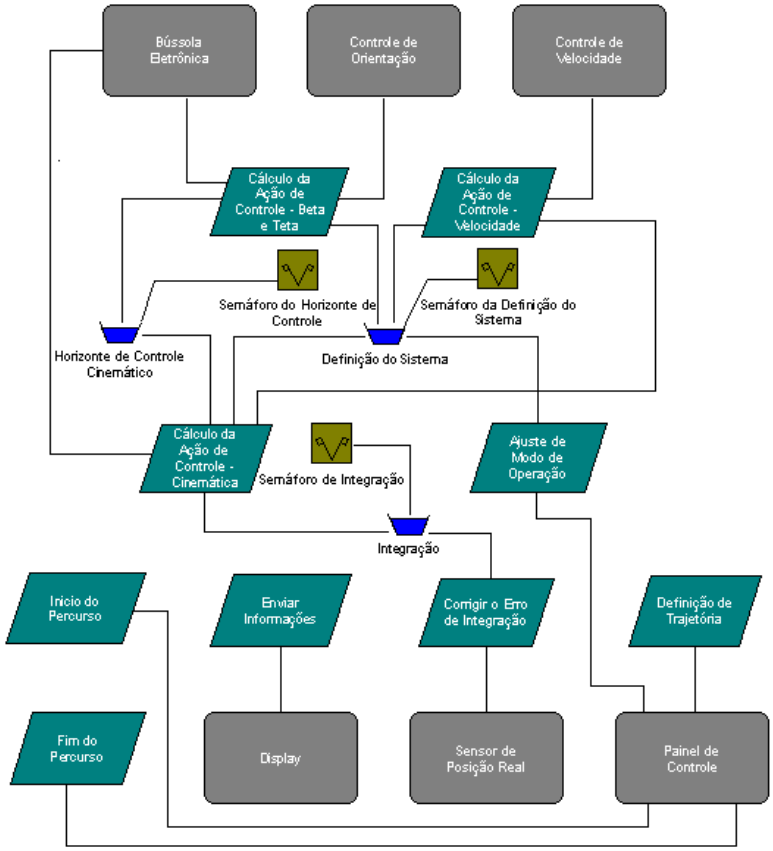

Figura 6.2: Recursos exclusivos utilizados pelas tarefas do SCST Completo com estrutura em cascata.

Neste caso, a tarefa de maior prioridade fica bloqueada neste recurso até que a tarefa de menor prioridade o libere ao final de sua seção crítica.

6.4 Modelagem UML do Sistema

Juntamente com a definição das restrições de tempo-real do sistema, é feita a modelagem deste utilizando a metodologia apresentada no Capítulo 5. A primeira etapa consiste na especificação dos casos de uso a partir da lista de funcionalidades. Em seguida, são identificadas as classes que compõem o sistema e a estrutura formada por suas associações. Com base nesta estrutura, são modeladas as interações necessárias à atender os casos de uso, bem como o comportamento dinâmico dos objetos. Por fim, são identificadas as tarefas do sistema e suas restrições temporais, de modo a expressá-las no modelo UML utilizando o perfil UML-TR.

6.4.1 Modelagem dos Casos de Uso

A partir das funcionalidades apresentadas na Seção 6.2, são modelados os casos de uso do SCST, que são apresentados no diagrama da Figura 6.3.

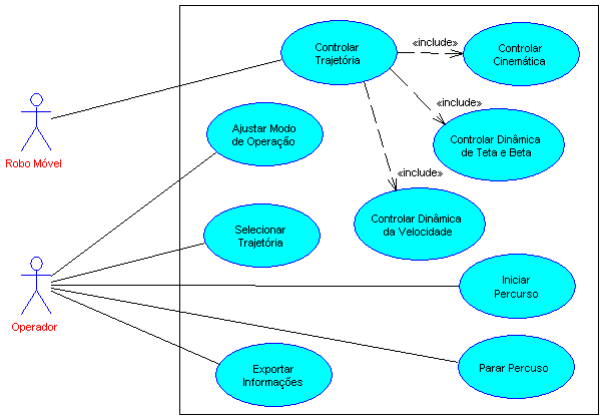

Figura 6.3: Casos de uso do sistema.

Os casos de uso *Iniciar Percurso* e *Parar Percurso* têm a função de controlar a operação do sistema, permitindo comandar o início e o fim do deslocamento do robô móvel. O caso de uso *Visualizar Informações* transmite a algum recurso de hardware, as informações referentes ao percurso. *Selecionar Trajetória* é o caso de uso que permite, através do banco de dados de trajetórias, definir a trajetória a ser percorrida. Já *Ajustar Modo de Operação* é o caso de uso utilizado que redefine a lei de controle de modo a atender diferentes critérios de desempenho. Por fim, *Controlar Trajetória* é o caso de uso responsável pela condução do veículo ou robô ao longo da trajetória pré-determinada, sendo dividido nos casos de uso específicos para o controle do modelo cinemático, do modelo dinâmico de β e $\dot{\theta}$ e do modelo dinâmico da velocidade v no centro de massa. O caso de uso *Controlar Trajetória* inclui ainda outros casos de uso relacionados a outras entidades externas além do ator *Robô Móvel*. Estas entidades são compostas pelos sensores e pelos atuadores do robô móvel, como mostra o diagrama de casos de uso da Figura 6.4.

Os casos de uso descritos na Figura 6.4 são, basicamente, operações destinadas a realizar o interfaceamento do SCST com os objetos físicos caracterizados pelos sistemas eletrônicos de sensoreamento e atuadores, situados no nível 1 da Figura 2.1.

Após a definição dos casos de uso do sistema, é realizada a modelagem das classes

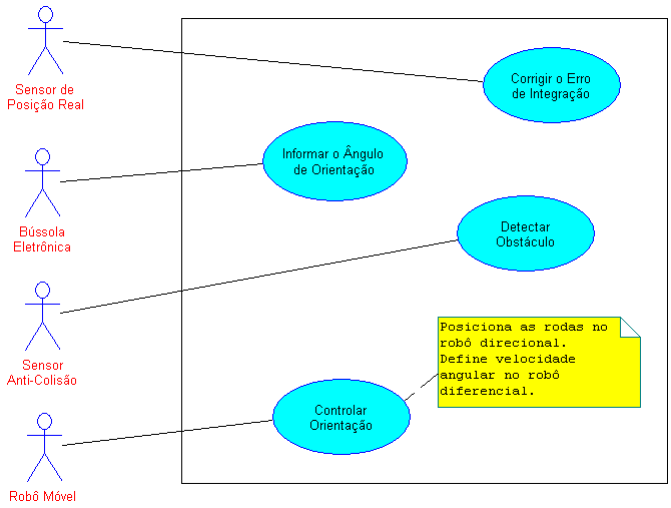

Figura 6.4: Casos de uso específicos dos sistemas de sensoreamento e atuadores.

e seus relacionamentos, com o objetivo de compor a estrutura lógica do sistema.

6.4.2 Modelagem das Classes

A próxima etapa do POO consiste na modelagem das classes do sistema. A identificação das classes foi feita com base nas principais entidades presentes nas estruturas representadas pelos diagramas de blocos das figuras 4.11, 4.17 e 4.18, entidades estas cujas operações foram detalhadas no Capítulo 4. São elas:

- **MPC:**

  Classe contendo controlador preditivo multivariável com base no algoritmo GPC, apresentado no Capítulo 4, e que caracteriza unidade de escalonamento representando a tarefa τ_1;

- **BetaThetaMPC:**

  Classe derivada de *MPC* especificamente para o controle da dinâmica de β e $\dot{\theta}$ para veículos direcionais, dadas pelas expressões (4.9) e (4.10) respectivamente. Representa a unidade de escalonamento da tarefa τ_8;

- **SpeedMPC:**

  Classe derivada de *MPC* especificamente para o controle da dinâmica da velocidade no centro de massa para veículos direcionais, dada pelas expressões (4.11, 4.12). Representa a unidade de escalonamento da tarefa τ_9;

- **ControlLawProcessor:**

  Classe com a parte do algoritmo GPC responsável pelo cálculo da lei de controle, com base no modelo do processo;

- **CommandProcessor:**

  Classe com a parte do algoritmo GPC responsável pelo cálculo da ação de controle, com base no modelo do processo;

- **Reference:**

  Classe para geração de trajetórias de referência com base no modelo cinemático de robôs diferenciais e direcionais. Oferece operações para geração de trajetórias circulares, quadrangulares, e nas formas de "oito"e "S";

- **PurePursuit:**

  Classe contendo o algoritmo *pure pursuit* de geração de trajetórias de aproximação. Conforme descrito no Capítulo 4, a geração de trajetórias de aproximação se faz necessária para validar o modelo linearizado utilizado no SCST;

- **SpeedController:**

  Interface para o sistema de controle de velocidade;

- **HeadingController:**

  Interface para o sistema de direção;

- **HeadingDriver:**

  Classe abstrata com características gerais para drivers de hardware de controle de orientação;

- **KDVAHeading:**

  Classe derivada de *HeadingDriver* contendo os drivers específicos para o controle de orientação e velocidade do robô KDVA (ver Capítulo 7);

- **MiniBajaHeading:**

  Classe derivada de *HeadingDriver* contendo os drivers específicos para o controle de direção do veículo Mini-Baja (ver Capítulo 7);

- **SpeedDriver**

  Classe abstrata com características gerais para drivers de hardware de controle de velocidade;

- **MiniBajaSpeed**

  Classe derivada de *SpeedDriver* contendo os drivers específicos para o controle de velocidade do veículo Mini-Baja (ver Capítulo 7);

- **Integrator:**

  Classe com método de integração numérica para determinar, em tempo-real, a posição do robô em coordenadas cartesianas no plano de deslocamento. Também caracteriza unidade de escalonamento representando a tarefa τ_4;

- **Sensor:**

  Classe abstrata com características gerais para sensores;

- **AngleSensor:**

  Classe derivada de *Sensor* que representa a bússola eletrônica;

- **RealPosSensor:**

  Classe derivada de *Sensor* que representa o sensor de posicionamento real;

- **CrashSensor:**

  Classe derivada de *Sensor* que representa o sensor anti-colisão;

- **SystemClock:**

  Relógio de tempo-real;

- **OperatorIO:**

  Interface com o operador, através da qual são realizados os comandos de início, fim e seleção da trajetória, ajuste de modo de operação e envio de informações, caracterizando portanto, a unidade de escalonamento responsável pelas tarefas não concorrentes J_2, J_3, J_5, J_6 e J_7.

Tendo sido determinadas as classes do sistema de controle, o próximo passo foi a identificação dos *atributos* e *operações* de cada classe e os seus relacionamentos. Os

atributos e *operações* foram definidos com base nos algoritmos GPC e *pure pursuit*, e demais métodos auxiliares necessários à técnica de seguimento de trajetória de veículos. Os relacionamentos entre as classes foram definidos separadamente para o controle baseado apenas na cinemática e para o controle em cascata, a partir dos diagramas de blocos das figuras 4.11, 4.17 e 4.18.

Seguimento de Trajetória com Modelo Cinemático

O diagrama de classes do SCST Completo baseado no modelo cinemático, conforme o diagrama da Figura 4.11, é apresentado na Figura 6.5.

A partir do diagrama da Figura 6.5, pode-se observar que as mensagens são quase todas enviadas pela classe *MPC*, que centraliza o controle das atividades, representando o bloco *Condução* do diagrama de blocos da Figura 4.11. Esta classe é composta pelas classes *ControlLawProcessor* que calcula a lei de controle, e *CommandProcessor* que calcula a ação de controle. Além de *MPC*, apenas as classes *SystemClock* e *OperatorIO* também realizam o envio de mensagens, todas destinadas a MPC. *OperatorIO* envia mensagens para a redefinição de parâmetros do sistema, enquanto que *SystemClock* envia mensagens periódicas invocando a execução do cálculo da ação de controle e da correção do erro de integração. Desta forma, com exceção das associações entre *MPC*, *OperatorIO* e *SystemClock* que são bidirecionais, todas as outras são unidirecionais no sentido de *MPC* para as demais classes. Quanto aos tipos de multiplicidade das associações, tem-se que todas são *1 para 1*, ou seja, no sistema de controle há apenas uma instância de cada objeto. A única exceção é a associação de composição da classe *OperatorIO*, que pode conter mais de um objeto da classe *Button*.

Na modelagem das classes, a opção por definir a trajetória de aproximação (*PurePursuit*) e o integrador (*Integrator*) como objetos distintos em vez de adicioná-los como objetos membros do controlador, tem como uma das razões, tornar facilitada a implementação do SCST utilizando outras técnicas de integração e geração de trajetória de aproximação. Neste caso, basta associar o controlador às classes contendo implementações destas técnicas. Se estas classes preservarem a mesma interface, as classes clientes (no caso, *MPC*) não precisam ser alteradas. A classe *PurePursuit* foi originada do subsistema *Gerador e Supervisor de Trajetória de Referência*, assim como a classe

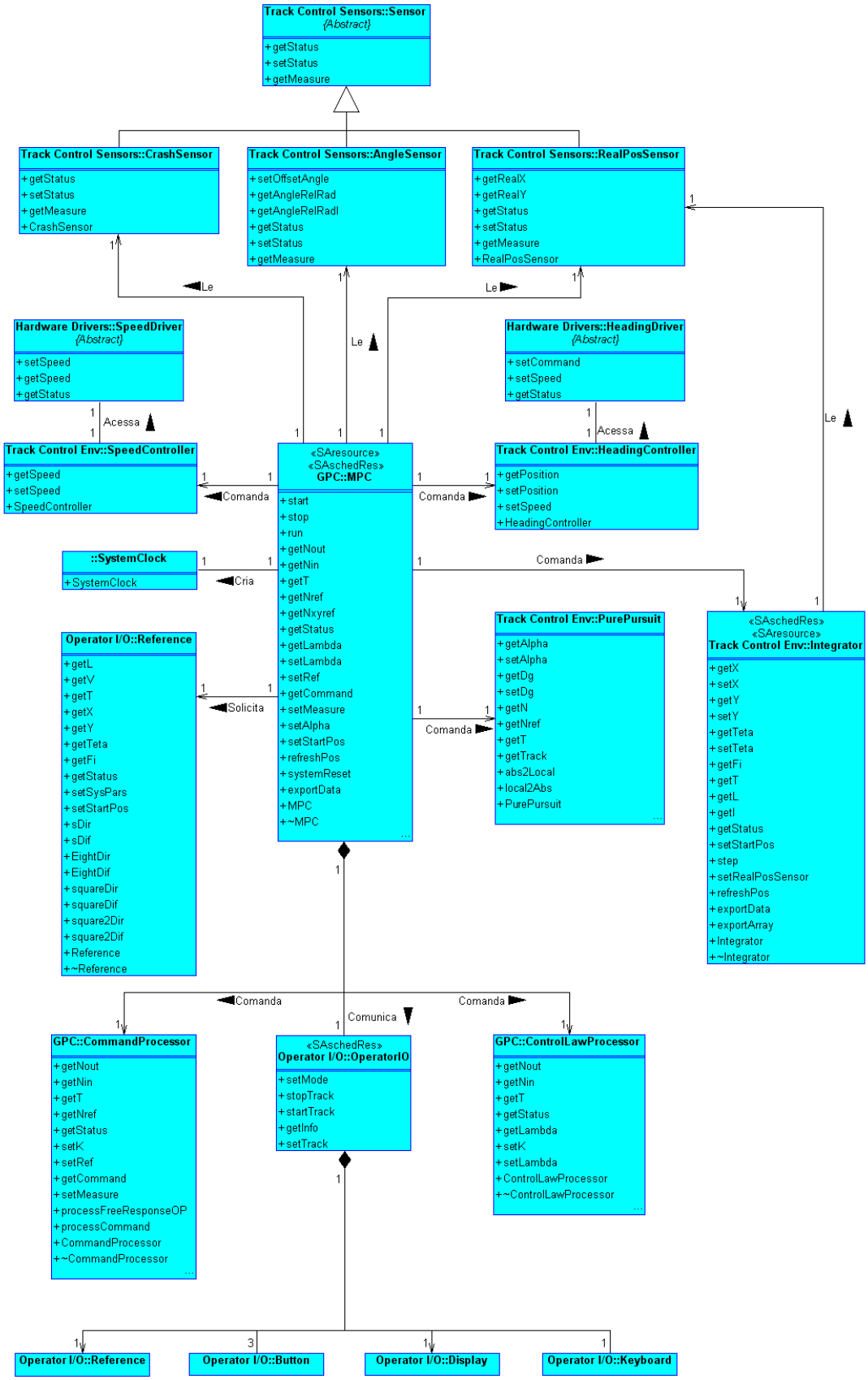

Figura 6.5: Diagrama de classes do SCST Completo baseado em modelo cinemático.

Reference, que foi originada do subsistema *Plano de Rota*, ambos pertencentes ao bloco *Planejamento de Rota* presente na Figura 4.11. A classe *Integrator* não é originada especificamente de nenhum bloco do diagrama da Figura 4.11, pois é considerada uma parte do subsistema *Gerador e Supervisor de Trajetória de Referência*. Da mesma forma, as classes que modelam os sensores são consideradas parte do bloco *Robô*.

Para realizar a comunicação do controlador com os sistemas eletrônicos que comandam as ações do robô móvel, foram modeladas as classes *HeadingController* e *SpeedController*, originadas do bloco *Sub-sistemas*, novamente da Figura 4.11. Estas classes oferecem operações de alto nível para comandar as ações de robôs diferenciais e direcionais a partir da associação com classes derivadas dos tipos abstratos *HeadingDriver* e *SpeedDriver*. Estas últimas, por sua vez, contêm os detalhes da implementação do código de acesso ao hardware específico de cada robô, de modo que estes detalhes ficam ocultos ao controlador.

Em relação aos requisitos de tempo-real, o uso do estereótipo *SAschedRes* nas classes *MPC*, *Integrator* e *OperatorIO* indicam que os objetos destas classes serão as unidades de escalonamento que executam as tarefas do sistema. As classes *MPC* e *Integrator* também são decoradas com o estereótipo *SAresource*, indicando que suas instâncias também agregam recursos compartilhados.

É importante destacar que embora o sistema de controle aqui projetado seja destinado à aplicações em robótica móvel, a classe *MPC* é um controlador preditivo multivariável de uso geral, ou seja, é aplicável a qualquer processo linear. Assim, de modo a promover a reutilização desta classe em outras aplicações, optou-se por não utilizar uma associação de composição da classe controladora com as demais classes auxiliares ao controle de trajetória, pois estas não são necessárias ao controlador em aplicações convencionais. Desta forma, os objetos das classes auxiliares são criados pelo controlador em tempo de execução apenas em aplicações de controle de trajetória.

Ainda nesta etapa, é modelado também o SCST Simplificado onde as definições da trajetória e do modo de operação são realizadas em tempo de compilação. Para isso, é criada a classe *MPCLite* que ao contrário da classe *MPC*, não agrega um objeto da classe *ControlLawProcessor* e não se associa à classe *Reference*. O diagrama de classes do SCST Simplificado pode ser visto na Figura 6.6.

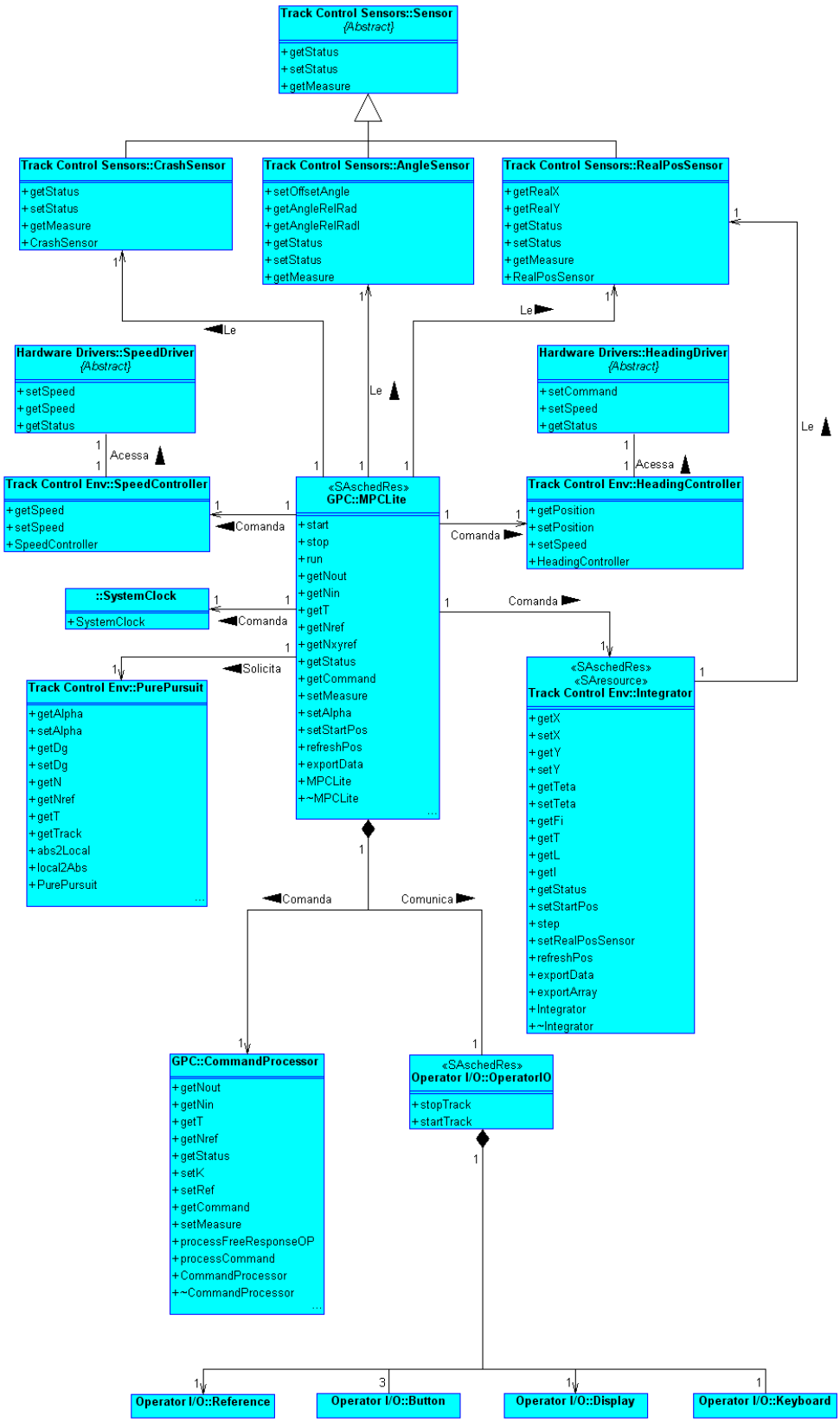

Figura 6.6: Diagrama de classes do SCST Simplificado baseado em modelo cinemático.

O diagrama de classes do sistema simplificado mostra que a classe *MPCLite* não apresenta o estereótipo *SAresource*, uma vez que a memória com as definições do sistema e da lei de controle é alocada e escrita em tempo de compilação, e por isso não é acessada de forma concorrente durante a execução.

Seguimento de Trajetória com Modelos Cinemático e Dinâmico

A representação do modelo de classes do SCST Completo é apresentada no diagrama da Figura 6.7. Diferentemente dos modelos das figuras 6.5 e 6.6, na estrutura em cascata a classe *MPC* não mais se associa às classes *HeadingController* e *SpeedController*, uma vez que a comunicação com os dispositivos de hardware passa a ser realizada através do nível da dinâmica, ou seja, o nível 2 da Figura 2.1. Este nível é representado pelas classes *BetaThetaMPC* e *SpeedMPC* que foram originadas respectivamente a partir dos dois subsistemas presentes no bloco *Controle da Dinâmica do Robô* das figuras 4.17 e 4.18.

Como mostrado nos diagramas de blocos das figuras 4.17 e 4.18, nas estruturas em cascata, a ação de controle calculada pelo controlador da cinemática serve de referência para o controlador da dinâmica de β e $\dot{\theta}$, que por sua vez envia o comando ao sistema de direção. Isso resulta nas associações unidirecionais entre *BetaThetaMPC* e *MPC* e *BetaThetaMPC* e *HeadingController* no diagrama da Figura 6.7. A primeira tem como finalidade permitir que *BetaThetaMPC* solicite à *MPC* as ações de controle do modelo cinemático para serem usadas como referência, e a segunda permite que a malha da dinâmica envie a ação de controle ao sistema de direção do veículo.

Já o controlador da dinâmica da velocidade v no centro de massa recebe a referência de *Reference*, que no caso representa o plano de rota, e atua no controlador de velocidade. Isso implica as associações unidirecionais entre *SpeedMPC* e *Reference*, e *SpeedMPC* e *SpeedController*.

A presença do esteriótipo *SAschedRes* nas classes *BetaThetaMPC* e *SpeedMPC* indica que estas representam as unidades concorrentes responsáveis pela execução das tarefas de controle da dinâmica do sistema. O uso do esteriótipo *SAresource* indica que as instâncias de *BetaThetaMPC* e *SpeedMPC*, assim como as de *MPC*, possuem

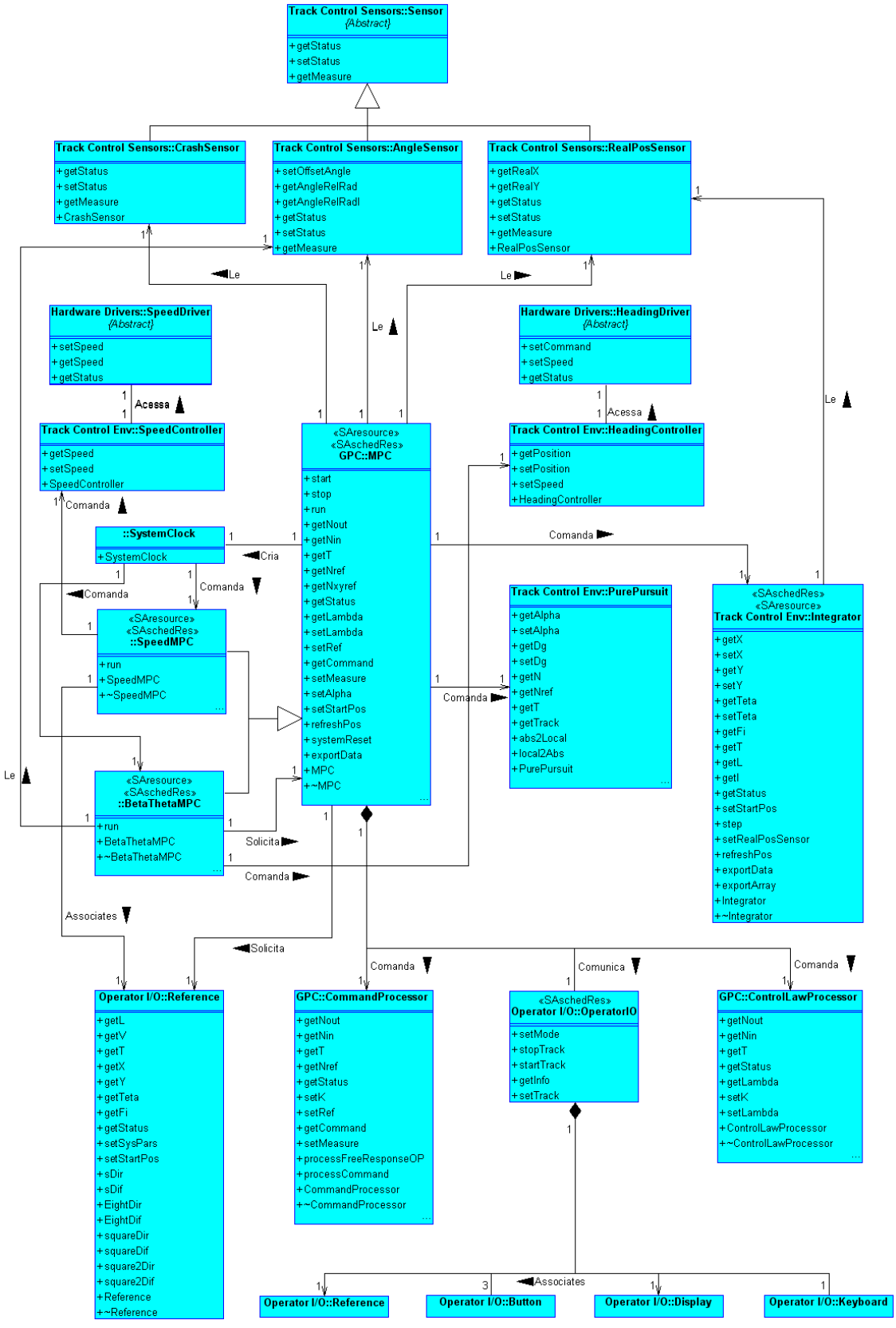

Figura 6.7: Diagrama de classes do SCST Completo baseado nos modelos cinemático e dinâmico.

recursos de acesso exclusivo, como as memórias da definição do modelo do processo e da lei de controle.

Embora a estrutura para o controle dos modelos cinemático e dinâmico possa ser implementada no modo simplificado, utilizando a classe *MPCLite* como controlador da cinemática e também como base para a derivação de *BetaThetaMPC* e *SpeedMPC*, pressupõe-se que o nível de exigência de uma aplicação de alto desempenho utilizando controle em cascata implique a utilização de hardware capaz de suportar a implementação em modo completo. Assim, a modelagem do SCST com estrutura em cascata é realizada apenas para este modo.

6.4.3 Pacotes

As classes desenvolvidas ao longo da modelagem foram organizadas em pacotes com funções específicas, de modo a promover a reutilização de software. As classes *MPC* e *MPCLite* e suas agregadas *ControlLawProcessor* e *CommandProcessor* foram organizadas no pacote *GPC*, que pode ser utilizado em aplicações genéricas de controle de processos. O ambiente necessário ao controle do comportamento cinemático com modelo linearizado em coordenadas locais motivou a criação do pacote *Track Control Env* com as classes *PurePursuit*, *Integrator*, *HeadingController* e *SpeedController*, que podem ser utilizadas junto a outros sistemas para tratar o problema de seguimento de trajetória. As classes que modelam os sensores foram agrupadas no pacote *Track Control Sensors*. Já as classes que modelam os drivers para acesso a hardware deram origem ao pacote *HardwareDrivers*. Por fim, a classe *OperatorIO* e suas agregadas foram agrupadas no pacote *Operator IO*. Os diagramas com os pacotes e os detalhes de cada classe são apresentados no Apêndice A.

6.4.4 Modelagem do Comportamento Interno

Na modelagem do comportamento interno, foram analisados os estados e transições a que estão sujeitos os objetos que compõem o sistema.

Os objetos das classes *Reference* e *PurePursuit* apresentam apenas o estado "ocioso"(*Idle*), sendo que após qualquer transição, estes voltam novamente para o mesmo

estado. Os eventos que ocasionam estas transições são mensagens do objeto da classe *MPC* solicitando os serviços oferecidos pelas operações destes objetos.

As classes *RealPosSensor*, *AngleSensor*, *CrashSensor*, *SpeedController* e *Heading-Controller* modelam objetos lógicos que servem como interface para a comunicação com os objetos físicos compostos pelos sistemas eletrônicos do robô móvel a ser controlado. Assim, também estes objetos apresentam apenas o estado "ocioso"a partir do qual ocorrem as transições em resposta as mensagens que solicitam os serviços das operações.

A modelagem do comportamento interno de um objeto da classe *MPC* e suas derivadas *BetaThetaMPC* e *SpeedMPC* é apresentada no diagrama de estados da Figura 6.8.

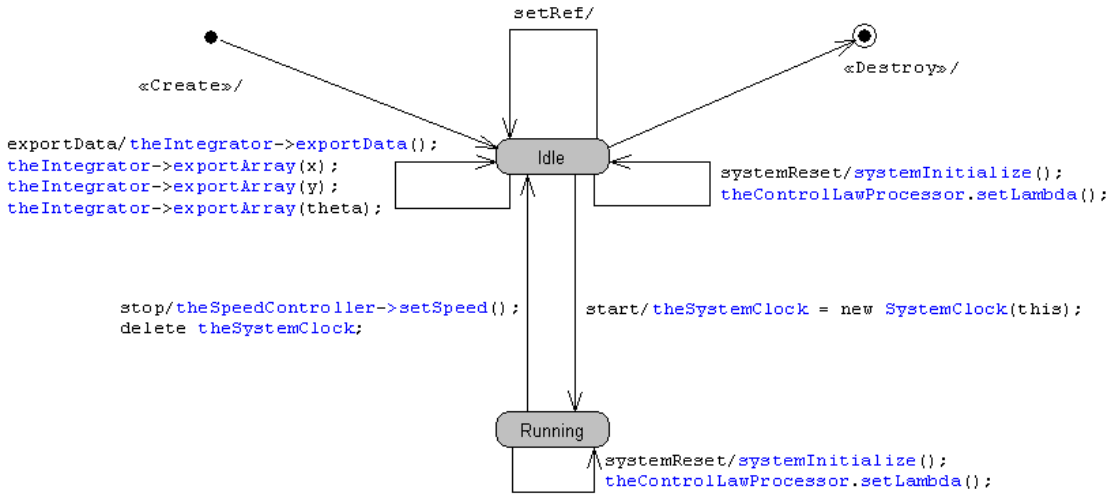

Figura 6.8: Diagrama de estados da classe *MPC* e suas derivadas *BetaThetaMPC* e *SpeedMPC*

Neste diagrama, pode-se identificar os estados "ocioso"(*idle*) e "executando"(*running*). Após a sua criação, o controlador permanece no estado "ocioso"até que seja invocada a operação *start()*, que comanda o início do percurso da trajetória. Ao iniciar o percurso, o controlador cria um objeto da classe *SystemClock* e muda seu estado para "executando". Uma vez "executando", o controlador recebe periodicamente mensagens do *SystemClock* solicitando o processamento da ação de controle. O controlador pode ainda, a partir do estado "executando", realizar uma transição para redefinir o modo de operação, retornando em seguida para o mesmo estado. Quando

no estado "ocioso", o controlador pode realizar transições para exportar os dados do percurso, para a redefinição do modo de operação ou para o registro da trajetória de referência, retornando em seguida para o mesmo estado. Já a transição do estado "executando"para o estado "ocioso"ocorre quando é ordenada a interrupção do percurso através de uma mensagem invocando a operação *stop()*. No caso do SCST Simplificado, o objeto da classe *MPCLite* e suas eventuais derivadas apresentam os mesmos estados, porém não existem as transições ocasionadas pelo ajuste do modo de operação e pela definição da trajetória.

Já a modelagem dos estados de um objeto da classe *Integrator* é apresentada na Figura 5.6, onde se percebe a presença dos estados "ocioso"(*Idle*) e "sobrecarregado"(*Overloaded*). Após instanciado, o integrador permanece no estado "ocioso", estando apto a realizar as suas operações. Estas operações podem ser o passo de integração, a exportação dos dados para arquivo, base de dados e/ou display entre outras, e a correção do erro de integração, que ocasionam transições iniciando e terminando no estado "ocioso". A transição para o estado "sobrecarregado"ocorre quando é utilizada toda a memória destinada ao armazenamento das coordenadas do deslocamento, ou seja, quando for solicitado um passo de integração após ser atingida a última posição do bloco de memória alocada. Ao atingir o estado "sobrecarregado", a integração é então desabilitada de modo a evitar a violação de outras áreas de memória.

6.4.5 Modelagem da Interação entre Objetos

Nesta seção é apresentada a modelagem das interações que ocorrem nos principais casos de uso apresentados no diagrama da Figura 6.3.

Controlar Trajetória

Este caso de uso consiste nas tarefas periódicas que são executadas no processo de condução do veículo. No caso do SCST baseado em modelo cinemático, a ação de controle é calculada apenas a partir do resultado do loop de controle do modelo cinemático. No caso do SCST para aplicações de alto desempenho, também os loop's de controle da dinâmica de β e $\dot{\theta}$ e da velocidade v no centro de massa se fazem necessários. Es-

tas tarefas são executadas em resposta a mensagens periódicas enviadas pela classe *SystemClock*, como mostra a Figura 6.9.

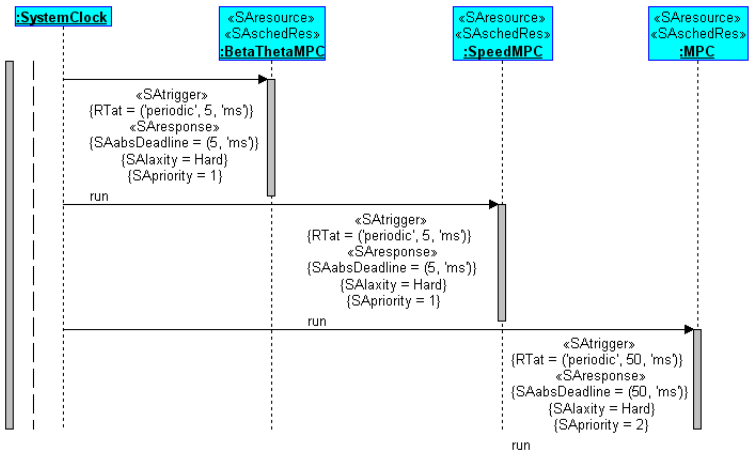

Figura 6.9: Diagrama de seqüência para seguimento de trajetória com estrutura em cascata.

A seguir são detalhadas as interações que ocorrem em cada uma das malhas de controle, para atender os casos de uso que compõem o caso de uso *Controlar Trajetória*.

Controlar Cinemática

As interações que ocorrem na execução do loop de controle do modelo cinemático também caracterizam a tarefa τ_1 e são modeladas no diagrama de seqüência da Figura 6.10.

Em resposta ao envio periódico de uma mensagem *run* pelo *SystemClock*, o controlador *MPC* dispara a execução de τ_1 que executa a seqüência de cálculo da ação de controle. O padrão periódico de ativação e as caracterísitcas de τ_1 são definidos no diagrama pelos estereótipos *SAtrigger* e *SAresponse*.

A primeira ação de τ_1 consiste no envio de uma mensagem ao *CrashSensor* para verificar a presença de obstáculo. A partir do resultado, é enviada uma mensagem ao *SpeedController* para definir a velocidade de deslocamento. Em caso da presença de obstáculo, o percurso é imediatamente interrompido setando-se velocidade zero. Caso contrário, é setada a velocidade desejada para o percurso.

Em seguida, é iniciado o cálculo da ação de controle, que, como foi visto, é feito com base na lei de controle, na predição das saídas e nos pontos da trajetória de apro-

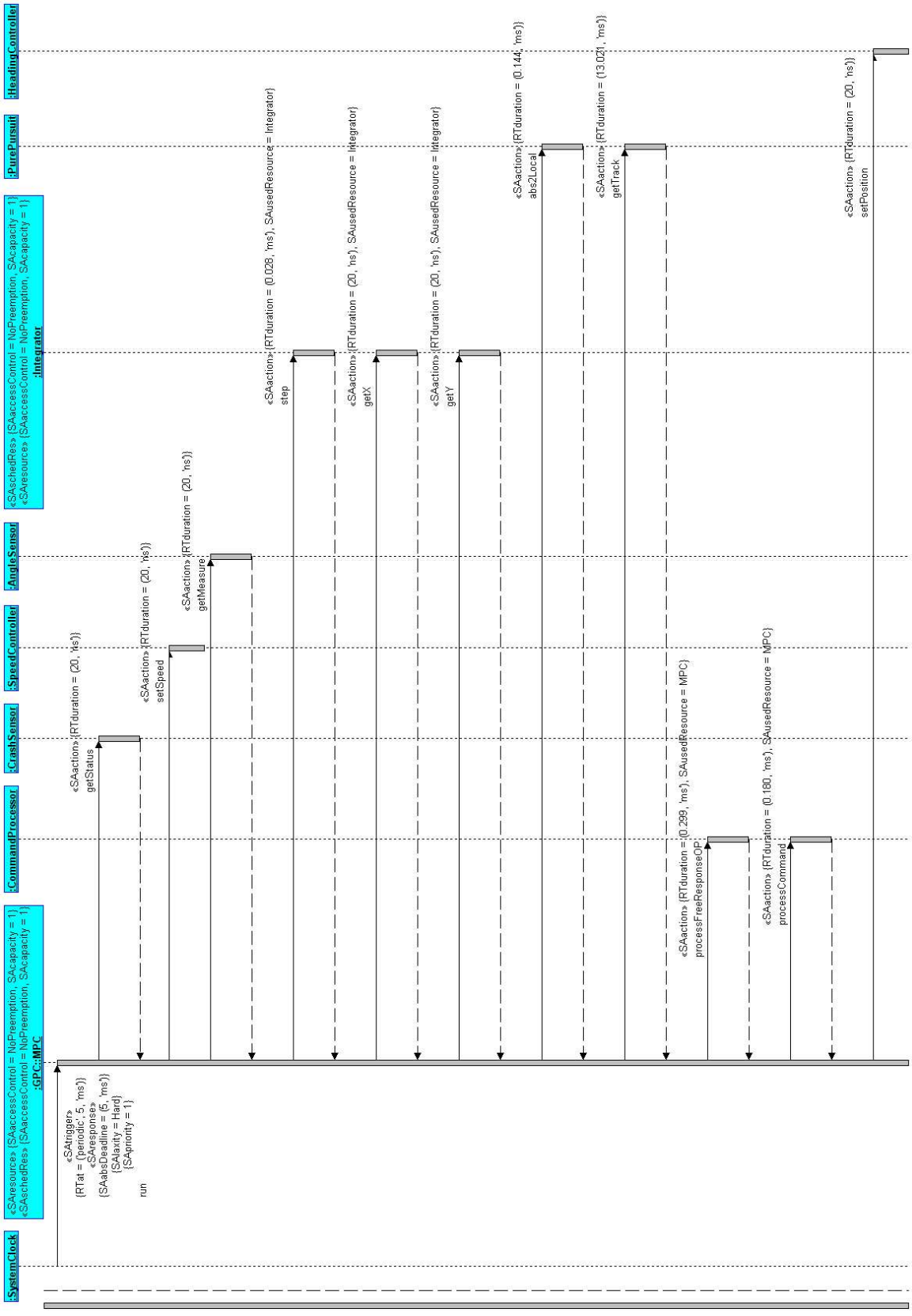

Figura 6.10: Diagrama de seqüência do loop de controle do modelo cinemático

ximação. Para realizar a predição das saídas do processo a ser controlado, os valores passados destas saídas devem ser obtidos através de medições sucessivas por meio de sistemas de sensoreamento. No caso do SCST, são necessários os valores passados do ângulo de orientação θ e da posição em coordenadas locais do robô. Assim, é enviada mensagem ao objeto da classe *AngleSensor*, solicitando a medida da bússola eletrônica. A medida de θ é então fornecida ao *Integrator* através do envio de mensagem que comanda o passo de integração, sendo em seguida solicitadas as coordenadas globais da posição do robô, com base na integração. O passo de integração e a solicitação das variáveis integradas são operações que fazem, respectivamente, leitura e escrita da memória de integração que, por sua vez, é um recurso compartilhado. A utilização deste recurso compartilhado nestas ações é representada pela marca *SAusedResource* definida como *"Integrator"* no estereótipo *SAaction*.

Obtidas as coordenadas integradas, a próxima ação consiste no envio de mensagem a *PurePursuit* solicitando a conversão destas coordenadas para o sistema de coordenadas locais, de modo que o controlador possa utilizá-las no modelo de predição.

A trajetória de aproximação em coordenadas locais é gerada na ação seguinte, com base na posição atual do robô e no ponto de destino sobre a trajetória de referência. Para isso, o controlador envia uma mensagem ao objeto da classe *PurePursuit* informando a posição atual do robô (em coordenadas locais), o valor de θ, a trajetória de referência (em coordenadas globais) e o valor do parâmetro *look-ahead*. O *PurePursuit*, por sua vez, determina a posição de destino sobre a trajetória de referência com base no *look-ahead*, converte esta posição para o sistema de coordenadas locais e gera a trajetória de aproximação, também em coordenadas locais, que então é retornada ao *MPC* em resposta a mensagem *getTrack*.

De posse da predição e da trajetória de aproximação, o objeto da classe *MPC* realiza o cálculo da ação de controle e, através de mensagem enviada ao objeto da classe *HeadingController*, aplica o comando ao sistema de direção do robô, finalizando o loop de controle. Pode-se observar que a mensagem enviada ao *HeadingController* é do tipo *assíncrona*, de modo que *MPC* permanece com o fluxo de controle. O esteriótipo *SAaction* que decora a operação de cálculo da ação de controle apresenta a marca *SAusedResource* definida como *"MPC"*, indicando o uso da memória referente

à lei de controle, que é agregada à classe *MPC*. A memória da lei de controle como recurso compartilhado da classe *MPC* é representada no diagrama da Figura 6.10 pelo estereótipo *SAresource* na classe *MPC*. As marcas *SAaccessControl*, definida como *"NoPreemption"*, e *SAcapacity*, definida com valor 1, representam a condição de acesso exclusivo e sem preempção.

A duração de cada uma das ações de τ_1 é definida no diagrama através da marca *RTduration* oferecida pelo esteriótipo *SAaction*. Todos os tempos apresentados no diagrama da Figura 6.10 são referentes a implementação em plataforma POWERPC, que é apresentada no próximo capítulo.

Controlar Dinâmica de β e $\dot{\theta}$

As interações que ocorrem na execução da tarefa τ_8, para o controle da dinâmica de β e $\dot{\theta}$, são modeladas no diagrama de seqüência da Figura 6.11.

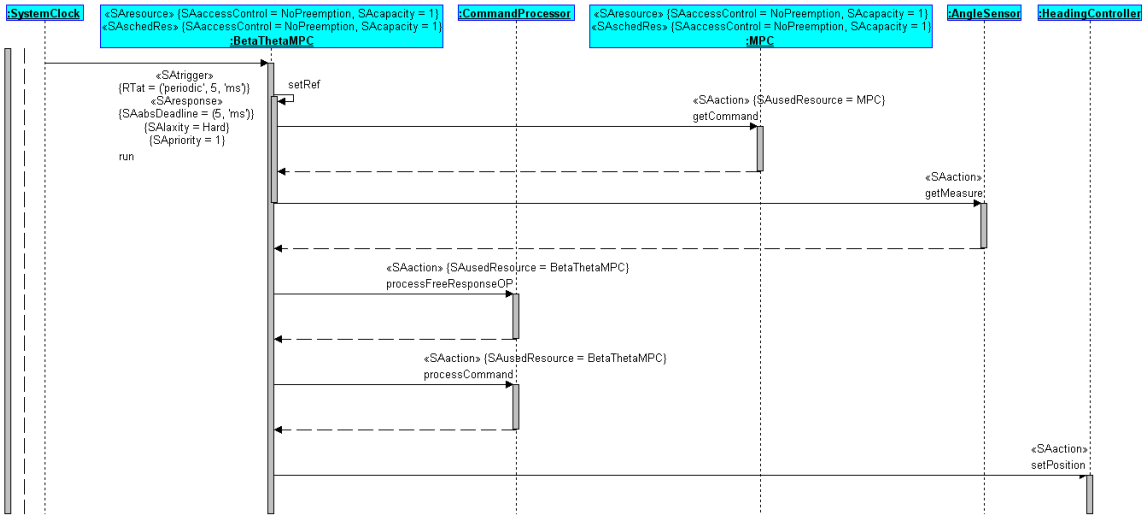

Figura 6.11: Diagrama de seqüência do loop de controle da dinâmica de β e $\dot{\theta}$.

Em resposta à mensagem periódica *run* enviada por *SystemClock*, o objeto da classe *BetaThetaMPC* inicia a seqüência de operações para o cálculo da ação de controle. Como primeira ação, *BetaThetaMPC* chama sua operação interna *setRef* que, por sua vez, envia uma mensagem ao controlador da cinemática solicitando a ação de controle para o modelo cinemático, que no caso é usada como referência. O estereótipo *SAaction* com a marca *SAusedResource* definida como *MPC* na mensagem *getCommand*, indica

que *BetaThetaMPC* utiliza a memória de *MPC* referente ao horizonte de controle. Esta memória, em aplicações com estrutura em cascata, é definida como um recurso exclusivo conforme definido na Seção 6.3.

Em seguida, é enviada a mensagem a *AngleSensor* solicitando a medição de θ. De posse da referência e da medição, são invocadas as operações *processFreeResponseOP* e *processCommand* para calcular a predição e a ação de controle respectivamente. Novamente aqui se pode perceber a marca *SAusedResource* definida como *BetaThetaMPC*, indicando o uso das áreas de memória referentes à definição do modelo dinâmico e da lei de controle. A ação de controle é então enviada ao sistema de direção por meio de mensagem à classe *HeadingController*.

No exemplo do diagrama da Figura 6.11, as marcas do estereótipo *SAtrigger* definem um período de amostragem dez vezes menor que a malha de controle da cinemática. Já as marcas do estereótipo *SAresponse* apresentam algumas características de τ_8, definidas na Seção 6.3, como nível máximo de prioridade e *deadline* do tipo *hard*.

Controlar Dinâmica da Velocidade no Centro de Massa

As interações que ocorrem no loop de controle da dinâmica da velocidade no centro de massa, caracterizadas pela tarefa τ_9, são modeladas no diagrama de seqüência da Figura 6.12.

Esta seqüência de interações é muito semelhante às interações para o controle da dinâmica de β e $\dot{\theta}$. Em resposta à mensagem periódica *run* de *SystemClock*, o objeto da classe *SpeedMPC* inicia a seqüência chamando *setRef* que desta vez, solicita a referência para um objeto da classe *Reference*. Já a medição da saída do processo é obtida por meio do controlador de velocidade *SpeedController*. Após a obtenção de referência e da medição, são calculadas a predição e a ação de controle que, por sua vez, é aplicada ao veículo por meio de *SpeedController*.

Os esteriótipos *SAtrigger* e *SAresponse* da Figura 6.12 mostram que as definições de τ_9 são as mesmas de τ_8, conforme definido na Seção 6.3. Já o uso da marca *SAusedResource* do esteriótipo *SAaction* nas operações *processFreeResponse* e *processCommand*, indica que estas fazem uso da memória que armazena o modelo do processo (no caso o a dinâmica da velocidade) e a lei de controle.

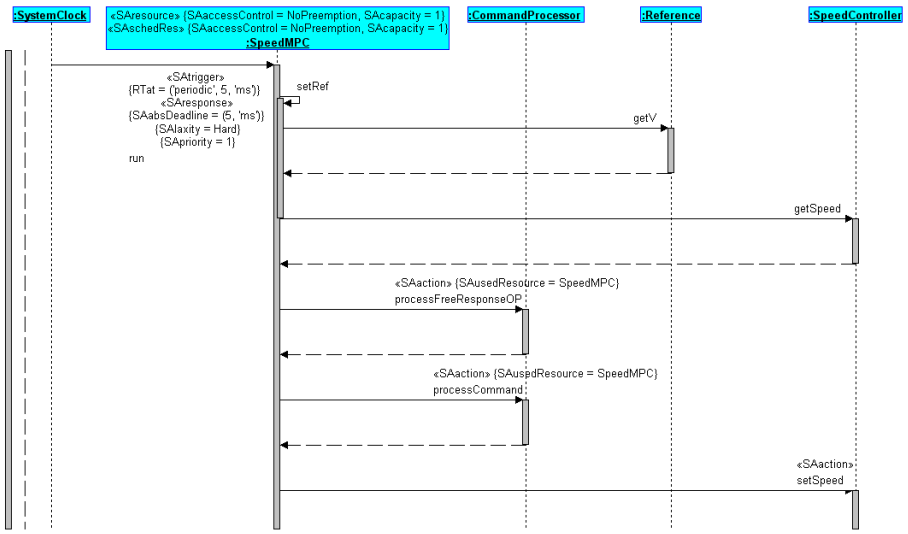

Figura 6.12: Diagrama de seqüência do loop de controle da dinâmica da velocidade no centro de massa.

Iniciar Percurso

Através deste caso de uso, o operador pode iniciar o percurso da trajetória pré-determinada. A Figura 6.13 apresenta o diagrama de seqüência com as interações envolvidas no comando de início de percurso.

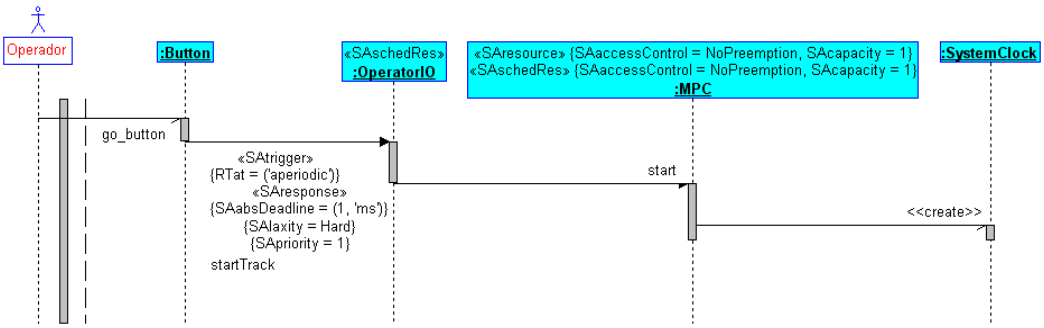

Figura 6.13: Diagrama de seqüência do comando de início de trajetória

Como mostra o diagrama de seqüência, a partir do evento *go button*, originado pelo comando do operador, o objeto da classe *Button* envia mensagem disparando em *OperatorIO* a execução da tarefa J_6, que consiste na ação de solicitar a *MPC* a criação de um objeto da classe *SystemClock*. Após instanciado, o objeto *SystemClock* passa a comandar periodicamente o cálculo da ação de controle, dando início ao percurso da trajetória.

Parar Percurso

Este caso de uso tem a função de permitir ao operador interromper o percurso da trajetória, como mostram as interações do diagrama de seqüência da Figura 6.14.

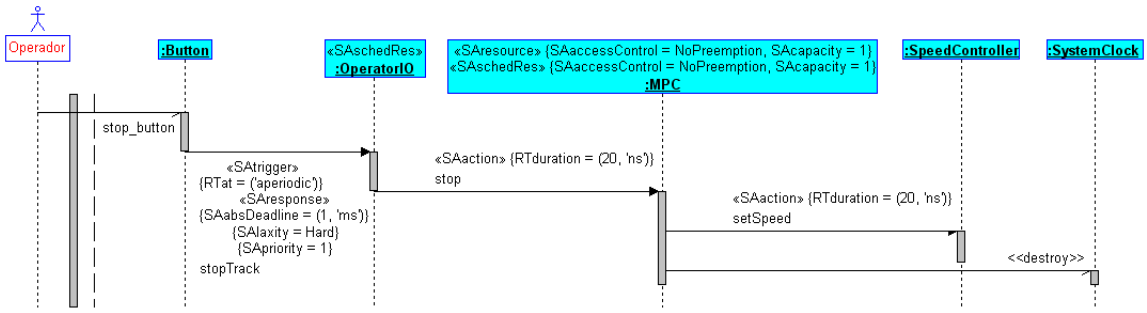

Figura 6.14: Diagrama de seqüência do comando de interrupção de trajetória

A partir do comando de parada imposto pelo operador, caracterizado pelo evento *stop button*, o objeto da classe *Button* envia mensagem disparando em *OperatorIO* a execução da tarefa J_7. Esta consiste no envio da mensagem *stop* a *MPC*, para parar o robô e destruir o objeto *SystemClock*.

Ajustar Modo de Operação

Caso de uso presente apenas no SCST Completo e com a finalidade de definir o modo de operação do SCST. As interações que ocorrem no ajuste do modo de operação do SCST baseado em modelo cinemático estão expressas no diagrama de seqüência da Figura 6.15.

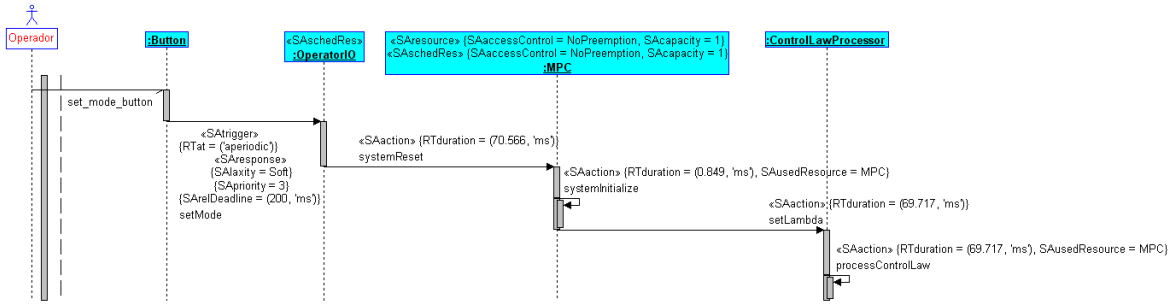

Figura 6.15: Diagrama de seqüência do comando de ajuste de modo de operação para SCST baseado em modelo cinemático.

A partir da solicitação de ajuste de modo, determinada pelo evento *set mode button*,

o objeto da classe *Button* envia mensagem disparando em *OperatorIO* a execução da tarefa J_2. Em J_2, *OperatorIO* envia mensagem a *MPC* invocando a operação de redefinição dos parâmetros do sistema. Esta operação por sua vez, invoca no objeto da classe *ControlLawProcessor* a operação de cálculo da lei de controle com base nas novas definições. As seções críticas de J_2 são representadas pela marca *SAusedResource*, definida como *"MPC"* no esteriótipo *SAaction*, decorando as mensagens que invocam as operações *systemInitialize* e *processControlLaw*. Esta última operação realiza a escrita da memória referente à lei de controle, enquanto que a primeira operação realiza a escrita da memória referente à definição do sistema. As características destes recursos compartilhados são descritas no estereótipo *SAresource* da classe *MPC*.

Também na Figura 6.15, os valores de tempo apresentados pelos esteriótipos *SAtrigger*, *SAresponde* e *SAaction* são referentes à implementação em POWERPC, apresentada no Capítulo 7.

No caso do SCST Completo com estrutura em cascata para controle dos modelos dinâmico e cinemático, o ajuste do modo de operação ocorre de modo semelhante, com *SystemClock* invocando a operação *systemReset* para cada um dos objetos controladores das classes *MPC*, *BetaThetaMPC* e *SpeedMPC*, como mostra a Figura 6.16.

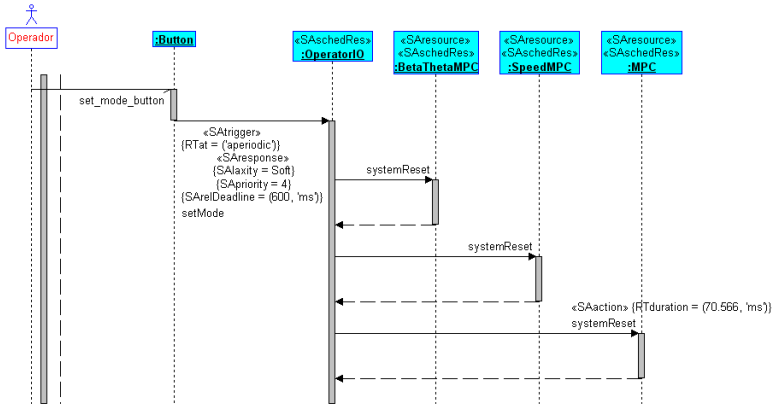

Figura 6.16: Diagrama de seqüência do comando de ajuste de modo de operação para SCST com estrutura em cascata.

Por questão de simplicidade, as operações realizadas por cada objeto controlador em resposta à mensagem *systemReset* foram omitidas no diagrama da Figura 6.16. Porém estas são as mesmas apresentadas na Figura 6.15.

Corrigir Erro de Integração

Este caso de uso tem a finalidade de corrigir periodicamente o erro de posicionamento que é acumulado ao longo do tempo, devido ao método de integração numérica utilizado para determinar em tempo-real, a posição do robô. Esta correção pode ou não ser necessária dependendo do nível de precisão desejado para a aplicação de destino. A correção do erro de integração implica a necessidade de um sensor capaz de informar a posição real do robô no plano de deslocamento. Embora este sensor não esteja disponível para a utilização neste projeto, as interações necessárias à utilização deste recurso foram modeladas e as operações necessárias à correção foram implementadas nas classes *MPC* e *Integrator*. A Figura 6.17 apresenta as interações necessárias a correção do erro de integração.

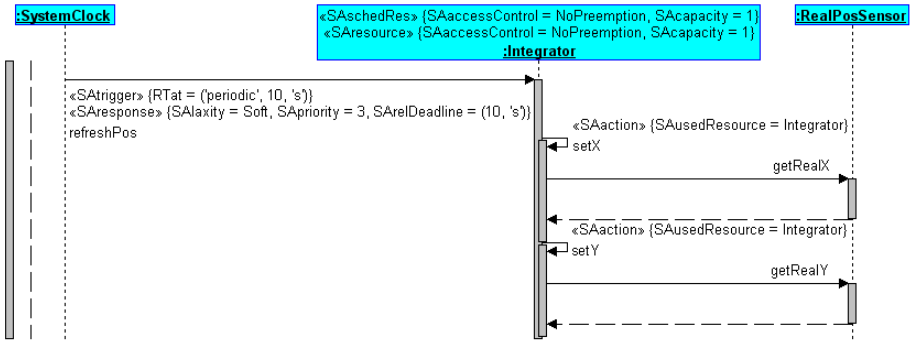

Figura 6.17: Diagrama de seqüência da correção do erro de integração

A correção do erro de integração se dá de forma periódica, por meio de mensagem enviada pelo *SystemClock*, que dispara em *Integrator* a execução da tarefa τ_4. Em resposta, o integrador envia mensagens ao sensor de posicionamento real, modelado pela classe *RealPosSensor*, de modo a obter as coordenadas da posição real. O acesso de τ_4 ao recurso compartilhado caracterizado pela memória de integração é representado no diagrama por meio da marca *SAusedResource*. A condição de recurso compartilhado da área de memória para a integração é representada na classe *Integrator* por meio do esteriótipo *SAresource*, noqual são utilizadas as marcas *SAaccessControl*, definida como *"NoPreemption"*, e *SAcapacity*, definida com o valor 1, para representar as necessidades de acesso exclusivo e sem preempção, definidas na Seção 6.3.

Enviar Informações

Este caso de uso foi modelado com a finalidade de proporcionar ao operador informações sobre as condições do robô ao longo do percurso. Como não havia a disponibilidade de dispositivos como displays ou monitores para o projeto, esta funcionalidade foi implementada de modo que as informações sejam exportadas ao final do percurso para arquivo ou para interface serial, dependendo da plataforma utilizada. A implementação atual do sistema realiza, ao final da trajetória, a exportação da evolução da posição e orientação do robô, sendo que as interações modeladas para esta atividade estão representadas no diagrama da Figura 6.18.

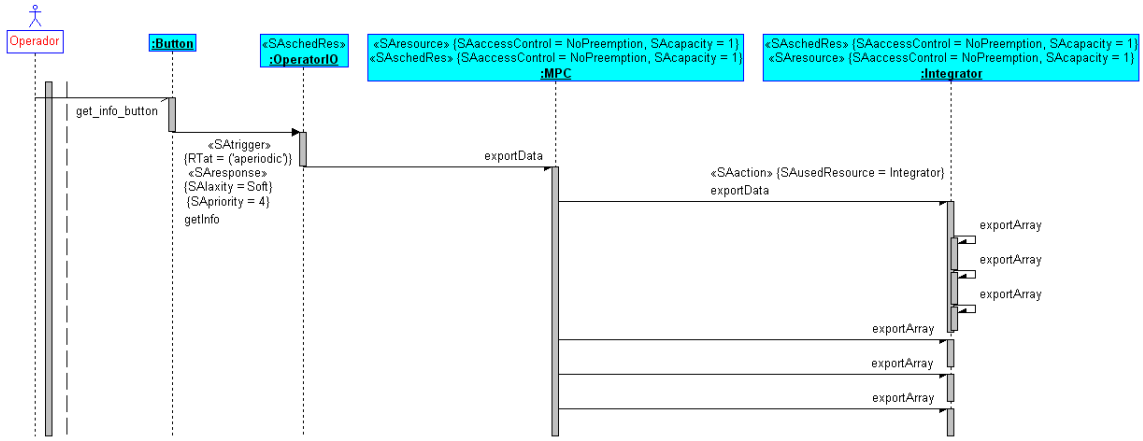

Figura 6.18: Diagrama de seqüência do envio de informações

Como mostra o diagrama da Figura 6.18, o acionamento de botão específico gera o evento *get info button*. Este evento faz com que o objeto da classe *Button* dispare a execução de J_5 na instância de *OperatorIO* através do envio da mensagem *getInfo*. Como resposta, a exportação dos dados é realizada através do envio de mensagens *exportData* e *exportArray* ao objeto da classe *Integrator* por parte do controlador *MPC*. Para realizar a exportação das coordenadas integradas, é realizada a leitura do recurso compartilhado caracterizado pela memória das coordenadas integradas. Este acesso é identificado pela marca *SAusedResource*, definida como *"Integrator"* no esteriótipo *SAaction* da ação que invoca a operação *exportData* de *Integrator*.

Em aplicações com maior restrição de recursos, este caso de uso pode ser dispensável.

Selecionar Trajetória

Caso de uso também presente apenas no SCST Completo e que permite a seleção da trajetória a ser percorrida a partir das opções oferecidas pela classe *Reference*. É importante observar que a seleção de trajetória não pode ser feita durante o deslocamento, de modo que o percurso precisa ser interrompido no caso de mudança de trajetória. As interações que ocorrem na seleção de trajetória são mostradas na Figura 6.19.

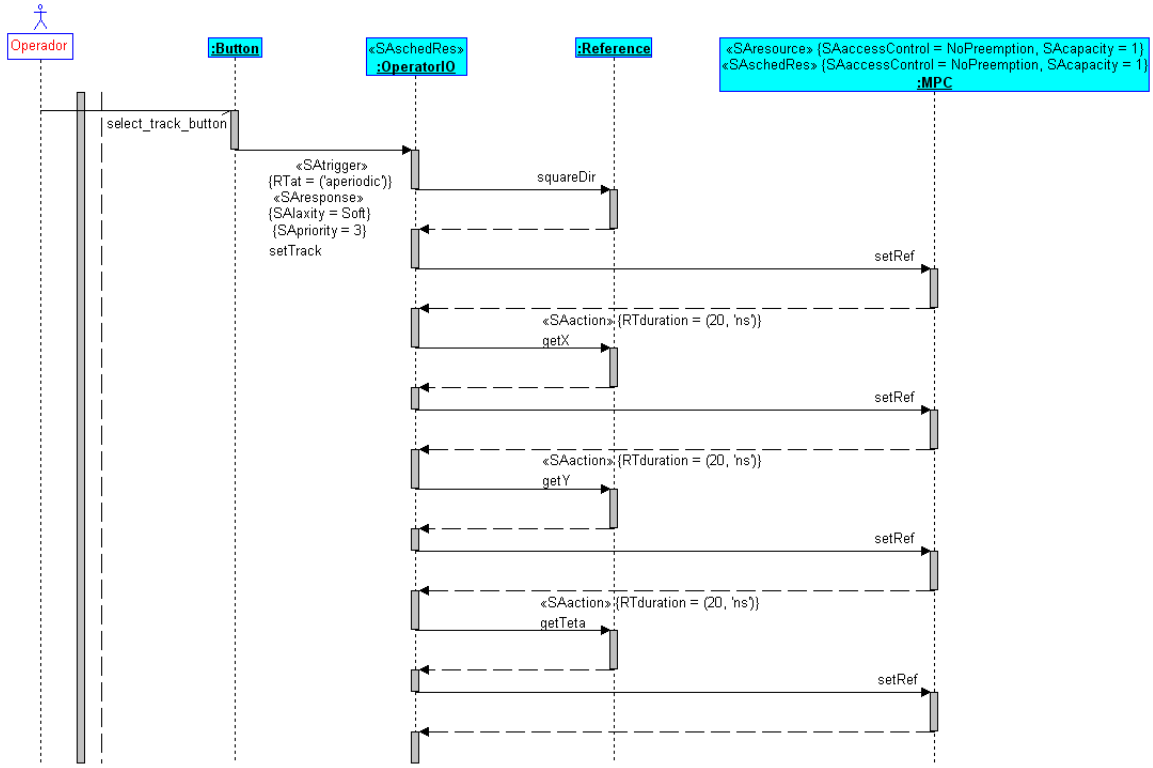

Figura 6.19: Diagrama de seqüência da seleção de trajetória.

Como mostra a figura, em resposta ao evento *select track button* originado por comando específico do operador, o objeto da classe *Button* dispara, em *OperatorIO*, a execução da tarefa J_5 pelo envio da mensagem *setTrack*. A resposta consiste na mensagem que invoca na instância de *Reference* a operação correspondente à trajetória desejada, que no exemplo do diagrama é em forma de quadrado. O objeto da classe *Reference* gera a trajetória e a armazena na memória disponível. Em seguida, *OperatorIO* envia mensagens solicitando referências a esta memória, para depois informá-las ao controlador *MPC*.

6.5 Algoritmo de Escalonamento

Em aplicações do SCST em que haja a necessidade de utilização de SOTR, a escolha do tipo de algoritmo de escalonamento ideal pode ser feita com base na análise das restrições de tempo-real presentes em cada modo de implementação.

6.5.1 SCST Simplificado

Em uma aplicação do SCST simplificado com base em modelo cinemático, na qual as configurações são executadas off-line e definidas no sistema em tempo de compilação, o conjunto de tarefas a ser executado durante o processo de condução do veículo é fixo e com instantes de ativação bem determinados, sendo composto por τ_1 ou τ_1 e τ_4, cujos parâmetros permanecem constantes. No caso da necessidade de SOTR para esta aplicação, um algoritmo escalonador do tipo off-line poderia suprir perfeitamente as necessidades do sistema, além de trazer a vantagem de um baixo custo computacional, pois, em tempo de execução, apenas as operações do *dispatcher* devem ser executadas. A principal vantagem do modelo com parâmetros fixos está na possibilidade de se utilizar complexos algoritmos de escalonamento de modo a buscar um solução ótima, pois toda a execução ocorre off-line. A Figura 6.20 apresenta um exemplo de escalonamento ao longo do tempo, das tarefas do sistema baseado em modelo cinemático.

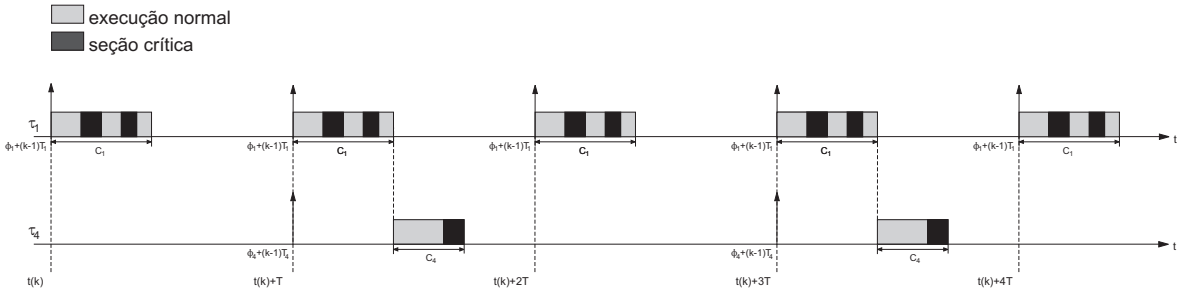

Figura 6.20: Exemplo de escalonamento das tarefas do SCST Simplificado baseado em modelo cinemático.

6.5.2 SCST Completo

Já no caso de uma aplicação em ambiente dinâmico, em que haja a necessidade de mudanças na configuração do sistema em tempo de execução, é também necessária

a execução da tarefa assíncrona J_2, tanto no controle baseado apenas no modelo cinemático quanto na estrutura em cascata.

Conforme comentado na Seção 6.3, como as tarefas τ_1, J_2, τ_4, τ_8 e τ_9 apresentam seções críticas que não podem ser interrompidas, o escalonador deve iniciar a execução de J_2 apenas em caso de garantia do cumprimento do *deadline* da tarefa τ_1 e, no caso do controle em cascata, também das tarefas τ_8 e τ_9. No caso de uma decisão baseada no WCET, tem-se que a tarefa J_2 pode eventualmente ser rejeitada desnecessariamente.

Também com relação a J_2, o fato desta poder apresentar um tempo de computação muito superior às demais tarefas implica a necessidade de um algoritmo preemptivo capaz de distribuir a sua execução ao longo de alguns períodos de amostragem, uma vez que suas seções críticas são pequenas e a execução pode ser interrompida de modo indeterminado durante os trechos de execução normal. A Figura 6.21 apresenta um exemplo de escalonamento das tarefas do sistema completo baseado em modelo cinemático, enquanto que a Figura 6.22 apresenta um exemplo de escalonamento das tarefas do sistema completo com estrutura em cascata.

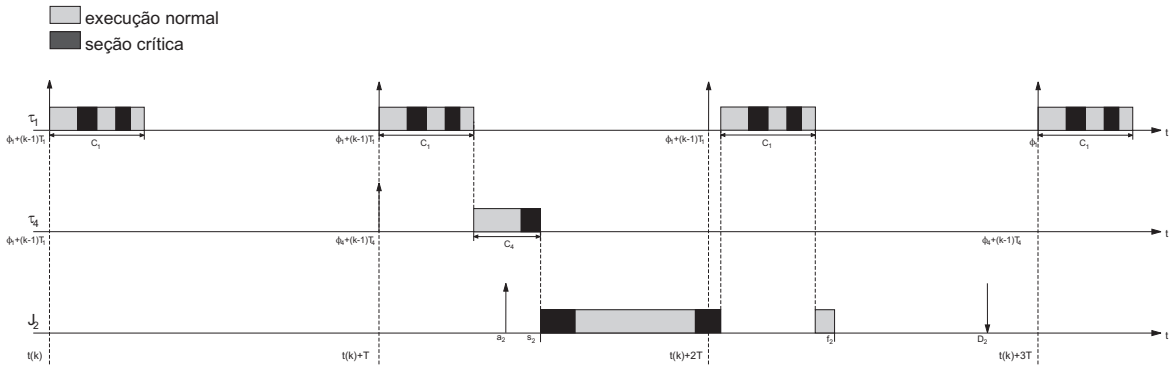

Figura 6.21: Exemplo de escalonamento das tarefas do SCST completo baseado e modelo cinemático.

Novamente no SCST Completo, um algoritmo de escalonamento off-line supriria as necessidades. Porém, devido às características de J_2, como tempo de execução elevado e acesso contínuo a recursos exclusivos, a opção por algoritmo on-line pode proporcionar uma execução mais otimizada, principalmente quando utilizada a estrutura em cascata.

Ainda em relação à estratégia de escalonamento, tem-se que algoritmos baseados em computação imprecisa podem ser aplicados ao sistema de controle de trajetória

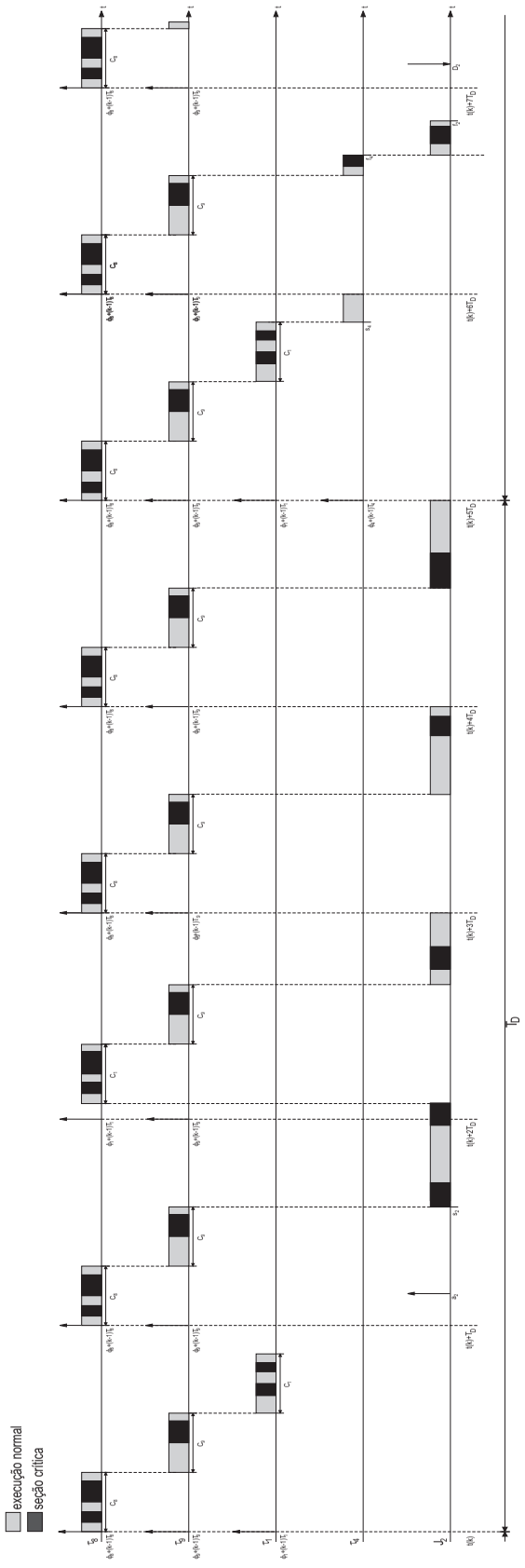

Figura 6.22: Exemplo de escalonamento das tarefas do SCST completo utilizando estrutura em cascata.

para tratar as tarefas τ_1 e τ_8. Como foi mencionado anteriormente, uma das ações de τ_1 e τ_8 consiste na medição do ângulo de orientação θ. Na análise das restrições de TR das tarefas, foi discutida a necessidade do tratamento de uma exceção para o caso da falha na leitura da bússola eletrônica. Neste caso, θ seria estimado através de integração, utilizando-se o modelo cinemático do veículo. Para tratar esta mesma questão no contexto da computação imprecisa, τ_1 e τ_8 podem ser divididas em M_1 e O_1, e M_8 e O_8 respectivamente, onde M_1 e M_8 utilizariam o valor estimado de θ e O_1 e O_8 substituiriam o valor estimado pelo valor real medido pela bússola eletrônica.

6.6 Conclusões

Este capítulo apresentou a modelagem do SCST utilizando conceitos de computação em tempo-real orientada a objetos, no qual foram identificados os principais componentes de software que compõem o sistema e as atividades que implementam os serviços necessários.

A partir da definição detalhada dos requisitos do problema, a linguagem UML foi utilizada para modelar os casos de uso do sistema, a estrutura de classes, as interações e o comportamento dinâmico dos objetos, destinados a atender cada um dos requisitos necessários. Através da identificação das restrições de temporais, foram obtidas informações sobre aspectos necessários à implementação do sistema de forma confiável, de modo a se obter comportamento previsível e garantia de execução em tempo-real. Foram definidos dois modos básicos de implementação, os modos simplificado e completo. O primeiro modo é destinado a aplicações em hardware mais restrito, nas quais as condições de operação são consideradas constantes e, portanto, são definidas em tempo de compilação. O segundo modo oferece toda configuração em tempo de execução e, permite assim, maior flexibilidade, em troca de maior custo computacional e utilização de memória, além da necessidade de SOTR em aplicações mais dinâmicas e em que se deseja comportamento otimizado.

Com relação à metodologia de projeto, pode-se perceber que a linguagem UML acompanhada do perfil UML-TR é bastante adequada à modelagem do comportamento do sistemas tempo-real orientados a objetos, permitindo expressar de forma clara o

comportamento dos objetos e as restrições temporais presentes nas atividades. A utilização da análise e projeto orientado a objeto é útil para orientar o cronograma de desenvolvimento do sistema de forma segura, direcionando as atividades no sentido da obtenção de código modular e de fácil manutenção e adaptação, possibilitando a reutilização dos componentes em diferentes aplicações de controle de processos.

Capítulo 7

Implementação do Sistema

7.1 Introdução

Este capítulo tem a finalidade de apresentar a implementação do SCST baseado em modelo cinemático a partir da modelagem realizada no Capítulo 6. Para validar o sistema de controle e verificar o seu comportamento em diferentes cenários, foram realizadas simulações e ensaios experimentais com implementações em diferentes plataformas e em diferentes modelos de veículos. Estas simulações e análises experimentais tem como objetivos verificar o desempenho do sistema de controle, coletar informações e parâmetros específicos referentes à execução do algoritmo em cada plataforma utilizada e analisar a influência de cada arquitetura no desempenho final. A avaliação do desempenho do sistema de controle é realizada através da comparação com os resultados simulados, e a avaliação da execução do algoritmo em plataformas embarcadas é realizada através da comparação com o a execução em plataforma X86.

Os ensaios experimentais foram realizados com implementações nas plataformas x86, DSP e POWERPC, sendo aplicados em veículos do tipo diferencial e direcional. Em algumas das implementações, o código do sistema de controle foi adaptado para atender as restrições do ambiente de desenvolvimento de cada plataforma. Da mesma maneira, para cada veículo testado, foram desenvolvidas camadas de "interface de aplicação de programa" (*Aplication Program Interface*, API) para acessar os subsistemas de cada veículo. Estas API's são utilizadas por classes denominadas *drivers*, destinadas à interação entre os diferentes níveis de controle de um veículo autônomo, conforme destacado na Figura 2.1. Com o uso destas classes é possível abstrair os detalhes de implementação dos subsistemas do veículo, que compreendem os sistemas de controle

e os sistemas de sensoreamento, tornando-os transparentes para os níveis de controle de dinâmica e condução do veículo, de modo que o código do SCST não precisa ser adapatado e assim não perde em generalidade.

Nas seções que seguem são descritos os detalhes de cada implementação realizada, sendo apresentados os resultados experimentais obtidos, bem como a comparação destes com os resultados teóricos das simulações, de modo a comprovar a eficiência do sistema tempo-real embarcado aqui desenvolvido.

7.2 Implementação no x86

A primeira plataforma em que foi implementado o algoritmo de controle foi a baseada em processadores da arquitetura X86. O principal motivo desta escolha foi o fato de que o sistema foi projetado e desenvolvido utilizando esta plataforma. A arquitetura X86 é muito utilizada em processos industriais, por meio de PCs industriais, que apresentam a mesma arquitetura dos PCs convencionais, porém utilizando apenas componentes de estado sólido. Assim, com a ausência de dispositivos como discos rígidos, drives de mídia removível e ventiladores, os PCs industriais podem operar em ambientes mais severos, suprindo as necessidades de muitas aplicações industriais que exijam alta carga computacional e armazenamento de grande volume de dados.

Entre as principais vantagens da plataforma X86 em relação a plataformas específicas para sistemas embarcados como DSPs e POWERPCs, pode-se citar que:

- É uma plataforma consagrada na indústria, sendo amplamente testada e utilizada em diversas aplicações e em diferentes áreas;
- Apresenta grande disponibilidade de periféricos e *device drivers*, proporcionando flexibilidade na escolha do hardware destinado a cada aplicação;
- Suporta uma maior variedade de sistemas operacionais de tempo-real;
- Apresenta maior variedade de linguagens de programação.

Estas características tornam a plataforma X86 adequada para suportar as aplicações mais exigentes no escopo de controle de seguimento de trajetória. Como desvantagem

para o uso desta arquitetura em sistemas embarcados, pode se citar o alto custo dos PCs industriais.

7.2.1 Programação do Algoritmo

A disponibilidade de memória e a capacidade de processamento da plataforma x86 possibilitam a implementação da versão completa do SCST. Para esta plataforma, a implementação foi realizada em linguagem C++. Apesar da opção por esta linguagem, é importante ressaltar que uma das grandes vantagens da arquitetura x86 é a diversidade de linguagens de programação disponíveis, permitindo codificação em diferentes níveis, de modo que outras linguagens e APIs também utilizadas em aplicações de TR podem ser utilizadas.

7.2.2 Determinação do Parâmetros das Tarefas

Para determinar os tempos de execução das tarefas τ_1 e J_2 definidas no Capítulo 6, foram feitos testes de execução do código do SCST em duas diferentes configurações da plataforma x86. A primeira configuração testada foi um processador Pentium 1 rodando a uma freqüência de clock de 166MHz com 32MBytes de memória RAM. A segunda configuração testada foi um Pentium 4 operando com um clock de 2.8GHz e 512MBytes de memória RAM. A Tabela 7.1 apresenta os tempos de execução obtidos nos testes com as duas configurações, utilizando os horizontes de controle N_u e predição N_2 iguais a 20:

Tarefa	**Descrição**	**Tempo de Computação C_i (ms)**	
		Pentium 1	Pentium 4
τ_1	cálculo da ação de controle	$1,950$	$0,094$
J_2	cálculo da lei de controle	$30,80$	$0,754$

Tabela 7.1: Tempos de execução na plataforma x86.

Estes resultados demonstram a capacidade da plataforma x86 em relação à execução do código do SCST e dão uma idéia da relação custo-performance entre os diferentes tipos de processadores desta família. Estes dados também são importantes

para estabelecer um comparativo com as outras plataformas implementadas, de modo a definir a adequação de cada uma para atender às necessidades de cada aplicação dentro da área de seguimento de trajetória para veículos autônomos. Com relação ao tempo de J_2, deve ser observado que este pode variar significativamente, influenciado pela forma como os dados referentes ao modelo do processo e os parâmetros do controlador são transmitidos para a memória da plataforma embarcada. Por motivo semelhante, o tempo de execução de τ_1 também pode ser influenciado pela forma como é realizada a comunicação com a bússola eletrônica, com o controle de velocidade e com o controle de direção.

As tarefas J_3 e J_5 não tiveram os seus tempos de execução determinados pois estes dependem diretamente do número de pontos da trajetória e do volume de informações a serem transmitidas. Também τ_4 não teve o seu tempo computado, pois este depende do tipo de sensor a ser utilizado e da forma como ele se comunica com a plataforma embarcada, além do fato de este sensor não estar disponível para os testes. Quanto aos tempos de J_6 e J_7, estes são desprezíveis em relação aos demais, uma vez que tais tarefas consistem basicamente em operações da alocação e recuperação de memória.

7.3 Implementação no DSP

Algumas aplicações na área de veículos autônomos não requerem todos os recursos da plataforma x86 e talvez não justifiquem a utilização desta. Nestes casos, procura-se utilizar plataformas mais específicas como os processadores digitais de sinais (*Digital Signal Processors*, DSP's), por exemplo, que, por serem mais específicas, apresentam custos reduzidos, mas com capacidade de processamento semelhante. Ao contrário dos microprocessadores, que são destinados a aplicações de caráter geral e normalmente executam softwares volumosos como sistemas operacionais, os DSPs são destinados a aplicações específicas em sistemas embarcados, apresentando características importantes, como baixo consumo de energia, dimensões reduzidas e alto poder de processamento, permitindo o desenvolvimento de dispositivos cada vez mais reduzidos e capazes de realizar tarefas complexas. Entre muitas aplicações de DSPs, pode-se citar telefonia móvel, instrumentação, controladores industriais, tratamento e reconhecimento de som

e imagem e sistemas de radar.

Para realizar a implementação do algoritmo de controle para seguimento de trajetória, foi utilizado um kit com o DSP 56F801 da Motorola, que apresenta uma arquitetura de 16 bits e opera a uma freqüência de clock de 60MHz. Este DSP é específico para aplicações de controle de motores, apresentando recursos de hardware muito importantes, como moduladores PWM, conversores analógico-digitais e timers, todos com grande flexibilidade de configuração. Maiores detalhes sobre a arquitetura e periféricos deste kit DSP são apresentadas no Apêndice B.

7.3.1 Programação do Algoritmo

Para realizar a programação do DSP, é fornecido um ambiente integrado de desenvolvimento composto pelas ferramentas *Code Warrior* e *Processor Expert*. O *Code Warrior* fornece ferramentas para compilação, depuração, gravação de memória flash e controle de versão entre outras, e permite a codificação em linguagens de alto nível como C e C++, além do tradicional Assembly. O *Processor Expert* é uma ferramenta gráfica para a configuração de modos de operação e periféricos, permitindo a configuração de timers, eventos e rotinas de tratamento de interrupções, comunicação serial entre outros, de forma bastante simplificada e sem a necessidade de conhecimento aprofundado dos detalhes de hardware. A partir das configurações definidas na interface gráfica é gerado todo código fonte necessário à implementação da configuração selecionada, restando ao projetista poucas tarefas além de adicionar o código referente ao algoritmo da aplicação. Outra vantagem do *Processor Expert* está na portabilidade do código, devido ao suporte deste a diferentes tipos de processadores da Motorola. A única restrição encontrada na utilização do *Processor Expert* foi o fato de que a versão disponível não dava suporte à linguagem C++, de modo que o algoritmo de controle teve que ser implementado em C. Como resultado prático, pode-se afirmar que esta ferramenta contribui reduzindo de maneira significativa o tempo de desenvolvimento das aplicações, uma vez que o tempo necessário para alterar a chamada das funções trigonométricas é pequeno se comparado ao tempo que seria necessário para estudar e configurar os periféricos de hardware, bem como os mecanismos de interrupção.

Após a análise da arquitetura do 56F801 e do ambiente de desenvolvimento, o código

do sistema de controle foi migrado para o kit do DSP. Devido à pouca memória disponível no kit utilizado, o sistema teve que ser implementado no modo simplificado ou seja, sem as tarefas J_2 e J_5 descritas no Capítulo 6. Assim, as definições do modelo do processo, do controlador e da trajetória devem ser processadas off-line e definidas em tempo de compilação. Outro detalhe importante na implementação do código foi o fato de que, na versão do Code Warrior para programação dos DSPs da família 56800, muitas funções matemáticas utilizadas no algoritmo, como funções trigonométricas e raiz quadrada, por exemplo, não estão implementadas na biblioteca *math.h*. De modo a otimizar o desempenho destas funções para a arquitetura do DSP, estas operações matemáticas são implementadas por código gerado pelo Processor Expert. Desta forma, foram necessárias algumas adaptações no código do algoritmo, pois no caso específico das funções trigonométricas, estas estão implementadas de maneira diferenciada. Como exemplo, pode-se tomar a função $atan(x)$, que está disponível como $atanOverPI(x)$, ou seja, o valor retornado é o arco tangente de x dividido por π.

7.3.2 Determinação dos Parâmetros das Tarefas

Tendo sido realizada a programação do algoritmo de controle no kit do DSP, foram feitos testes de execução em bancada, de modo a quantificar os parâmetros de TR das tarefas implementadas. Os tempos de computação obtidos nas medições com os horizontes de predição e controle iguais a 20 podem ser vistos na Tabela 7.2:

Tarefa	**Descrição**	**Tempo de Computação C_i (ms)**
τ_1	cálculo da ação de controle	$0,338$

Tabela 7.2: Tempos de computação na plataforma DSP.

Como já foi mencionado, as tarefas J_3, τ_4, J_5, J_6 e J_7 não tiveram seus tempos determinados pelos mesmos motivos descritos na Seção 7.2. Já a tarefa J_2, como foi mencionado, não foi implementada nesta plataforma e por isso não tive seu tempo determinado. A partir do resultado da Tabela 7.2, pode-se observar que o DSP 56F801 supre muito bem as necessidades da aplicação no que diz respeito à capacidade de processamento, pois os tempos de execução das tarefas se mostraram rápidos o suficiente

para controlar veículos dentro dos limites de velocidade propostos, sendo comparáveis aos tempos obtidos com a plataforma x86. Quanto às restrições impostas pela capacidade de memória, tem-se que estas ocorrem em função da limitação específica do kit DSP utilizado, de modo que o sistema pode ser implementado de forma completa em uma opção de hardware em que o processador tenha mais memória disponível.

7.4 Implementação no PowerPC

Apesar de ser um processador, e não um microcontrolador, o POWERPC vem sendo bastante utilizado em sistemas embarcados. Ele une um baixo consumo com um alto poder de processamento. Os POWERPCs são muito utilizados em controladores industriais, *gateways*, instrumentação, sistemas de reconhecimento de som e imagem, sistemas para a indústria automotiva e sistemas de telecomunicações, entre outras aplicações.

A implementação do algoritmo de controle nesta plataforma foi realizada utilizando-se o kit LITE5200 da Freescale, que utiliza o POWERPC MPC5200 da Motorola, um processador de 32 bits que opera a uma freqüência de clock de 400MHz. O kit também dispõe de 16Mbytes de memória *flash*, 64Mbytes de memória RAM e diferentes tipos de periféricos de comunicação. Com a presença de todas estas características, percebe-se a grande capacidade desta plataforma, o que permite a implementação das diversas funcionalidades discutidas no Capítulo 6, de modo a atender as necessidades das aplicações mais exigentes dentro da proposta de controle de trajetória. Em função da presença de todos estes recursos de hardware, esta plataforma foi a escolhida para a realização dos ensaios experimentais no veículo Mini-Baja, pois a boa disponibilidade de memória permite o armazenamento dos dados do percurso e as interfaces de comunicação possibilitam a coleta destes dados para análise.

7.4.1 Programação do Algoritmo

A programação do kit, assim como no caso do DSP, também é feita pela ferramenta *Code Warrior* em um versão específica para esta plataforma. Porém, para esta versão do *Code Warrior*, não está disponível o *Processor Expert*, e sim uma outra ferramenta chamada *MPC Quick Start*, que consiste em uma ferramenta gráfica para a habilitação,

desabilitação e seleção de modos de operação de periféricos. Embora seja uma interface muito boa entre o programador e os registradores de configuração de hardware, o *MPC Quick Start* não é uma ferramenta auxiliar tão poderosa se comparada ao *Processor Expert*, pois o código gerado apenas define o modo como o hardware e periféricos vão operar, de modo que ajustes, como temporização de timers, definição de eventos e rotinas de tratamento de interrupções e rotinas de acesso a periféricos, devem ser implementadas pelo projetista. Da mesma forma que o ambiente de desenvolvimento disponível para o DSP, o *MPC Quick Start* permite apenas a codificação em linguagem C, embora o *Code Warrior* aceite também programação em C++. Já uma vantagem do ambiente de desenvolvimento para POWERPC está no fato das bibliotecas padrões serem mais completas, a exemplo da biblioteca *math.h*, que contem todas as funções matemáticas necessárias à implementação do algoritmo.

Tendo sido analisadas a arquitetura do POWERPC e as características do ambiente de desenvolvimento, foi realizada a programação do algoritmo de controle utilizando o código desenvolvido em linguagem C. Devido à grande disponibilidade de memória do kit, foi possível implementar o sistema completo modelado no Capítulo 6. A disponibilidade de todas as funções matemáticas também permitiu implementação do código praticamente sem alterações. Na realidade, a única alteração necessária foi devido ao fato de que as rotinas de alocação dinâmica de memória estavam na biblioteca *stdlib.h* em vez da biblioteca *malloc.h*, bastando alterar as diretivas do pré-processador nos arquivos que incluem *malloc.h*.

7.4.2 Determinação dos Parâmetros das Tarefas

Os mesmos testes de execução realizados em bancada para quantificar os parâmetros de TR das tarefas nas plataformas anteriores, foram também realizados com código aplicado à plataforma POWERPC. Os tempos de execução obtidos nas medições e utilizando-se os horizontes iguais a 20, podem ser vistos na Tabela 7.3:

Como esperado pode-se observar, a partir dos resultados obtidos, que o POWERPC atende muito bem às necessidades das aplicações propostas e, como a plataforma X86, apresenta recursos capazes de suprir as necessidades de aplicações mais elaboradas,

Tarefa	Descrição	Tempo de Execução C_i (ms)
τ_1	cálculo da ação de controle	$13,686$
J_2	cálculo da lei de controle	$70,566$

Tabela 7.3: Tempos de computação na plataforma POWERPC.

como, por exemplo, a integração em sistemas distribuídos e comunicação com sistemas integrados de controle de tráfego, que se encontram no topo da hierarquia apresentada na Figura 2.1. Também aqui as tarefas J_3, τ_4, J_5, J_6 e J_7 não tiveram seus tempos determinados pelos mesmos motivos descritos na Seção 7.2.

7.5 Avaliação das Plataformas

De modo a se obter uma avaliação do desempenho de cada plataforma na execução do algoritmo de controle, é realizada uma comparação das características dos tempos de execução, tomando como referência a plataforma X86. A comparação destas características tem como objetivo determinar a plataforma mais adequada às necessidades de cada tipo de aplicação.

Os testes realizados com processadores Pentium I e Pentium 4 mostram a grande capacidade da plataforma X86 na execução do algoritmo de controle, o que pode ser observado na Tabela 7.1. Outra grande vantagem da plataforma X86 é a versatilidade proporcionada pela grande variedade de periféricos, que permitem a integração com diferentes arquiteturas. Como desvantagem, podem ser citados os fatos de que a estrutura de hardware dos sistemas X86 convencionais não são apropriadas para sistemas embarcados e as estruturas destinadas a aplicações industriais, como os chamados PCs industriais, normalmente são soluções de alto custo.

Os resultados da execução do algoritmo no DSP mostraram que esta plataforma também tem grande capacidade de processamento, sendo capaz de executar o código do algoritmo de controle em tempo comparável aos processadores mais poderosos da arquitetura X86, o que pode ser visto ao se coparar os dados da Tabela 7.2 com os da Tabela 7.1. Outras vantagens do DSP são as dimensões e consumo reduzidos, o que o torna adequado para sistemas embarcados. Quanto às desvantagens, tem-se a pouca

quantidade de memória, o que limita a utilização a aplicações mais específicas, nas quais as condições de operação sofrem poucas variações.

Da mesma maneira que as plataformas anteriores, a plataforma POWERPC também apresentou bom desempenho na execução do código, como mostra a Tabela 7.3. O POWERPC apresenta grande volume de memória e boa variedade de periféricos de comunicação, assim como o X86, e também apresenta dimensões reduzidas e baixo consumo, assim como o DSP, sendo apropriado para aplicações em sistemas embarcados. Porém, devido a estas caracterísitcas, o POWERPC também pode ser uma solução cara em algumas aplicações.

Assim, no caso de aplicações com robôs móveis de pequeno porte, em baixas velocidades e com condições de operação que sofrem pouca variação, a plataforma DSP é naturalmente a mais adequada, pois é a que apresenta o menor custo, o menor consumo e também as menores dimensões, se comparada às plataformas X86 e POWERPC. Já em aplicações de alto desempenho, nas quais a trajetória, a velocidade e a carga transportada possam variar significativamente, maior capacidade de comunicação e volume de memória são necessários, além do poder de processamento. Desta forma, as plataformas X86 e POWERPC passam a ser as mais adequadas, sendo que a opção entre uma e outra pode ser dada em função de necessidades específicas da aplicação em questão, como por exemplo, tipo de arquitetura de comunicação, armazenamento de dados ou custo.

7.6 Aplicação a um AGV Diferencial

Para verificar o desempenho do SCST com modelo cinemático aplicado a um veículo diferencial, foram realizados ensaios experimentais utilizando-se o veículo diferencial *Kit de Desenvolvimento para Veículos Autônomos*, ou KDVA, desenvolvido em FRANZON (2004) e apresentado na Figura 7.1.

Este veículo é movido por motores elétricos, sendo capaz de se deslocar com velocidades de até 0.4 m/s. O acionamento das rodas é feito por modulação PWM e a leitura de velocidade destas é feita por meio de tacômetros. O veículo possui ainda uma bússola eletrônica para medir o ângulo de orientação e um sensor de cabo-guia,

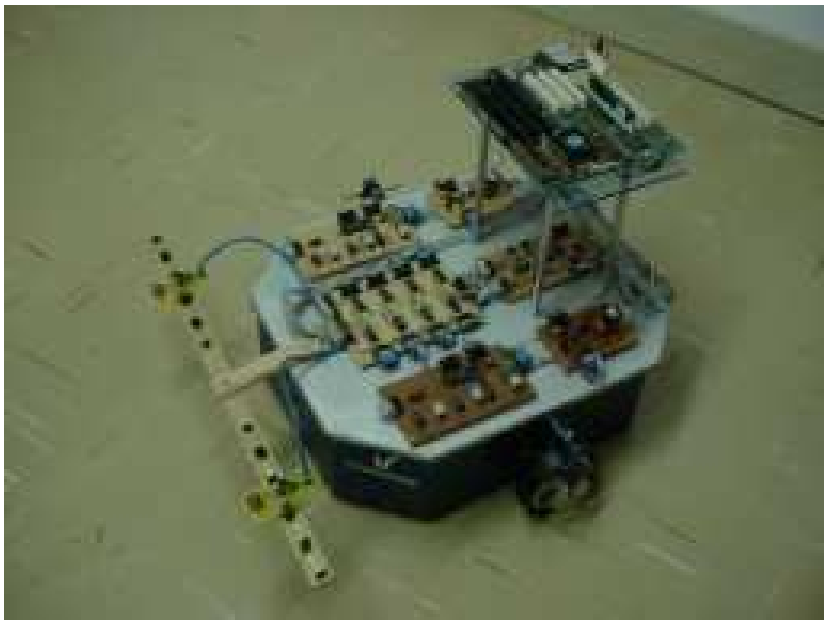

Figura 7.1: Veículo KDVA.

que permite medir o afastamento perpendicular em relação à tangente da trajetória. Estes subsistemas de acionamento e medição são comandados por uma CPU composta por um processador *Pentium I* operando em uma placa-mãe embarcada ao veículo. A configuração de hardware desta CPU é igual à utilizada para a coleta dos dados da Tabela 7.1 e a comunicação com os subsistemas é feita por meio da porta paralela. A energia do veículo é fornecida por meio de duas baterias de 12V, uma para alimentar a eletrônica de potência e a outra para alimentar a CPU, por meio de uma fonte ATX adaptada. Maiores detalhes sobre o veículo KDVA são apresentados em FRANZON (2004).

Para a realização dos ensaios, foram necessários o projeto e a implementação do primeiro nível da hierarquia representada pela Figura 2.1, representado por um subsistema de controle de orientação baseado na estrutura diferencial, que permita aplicar, de maneira correta, as ações de controle dimensionadas pelo SCST. Para isso, o primeiro passo foi projetar e implementar sub-malhas de controle de velocidade para cada uma das rodas, utilizando os moduladores PWM, os tacômetros e a CPU disponíveis no hardware embarcado. Com estes recursos, foram projetados dois controladores PI digitais independentes para impor aos motores a velocidade de rotação desejada.

Após a implementação destas sub-malhas de controle, foi desenvolvida uma API de

alto nível para controle de orientação, que utiliza estas sub-malhas para impor a velocidade individual de cada roda de modo a produzir o movimento desejado. Esta API é utilizada pela classe *KDVAHeading*, implementada a partir da derivação da classe abstrata *HeadingDriver*, que consiste em um driver de alto nível para o comando das rodas do KDVA. A partir da associação com a classe *KDVAHeading*, a classe *HeadingController* pode comandar o robô KDVA sem qualquer conhecimento dos detalhes de hardware. O diagrama de colaboração da Figura 7.2 apresenta a estrutura lógica da implementação do SCST aplicada ao KDVA.

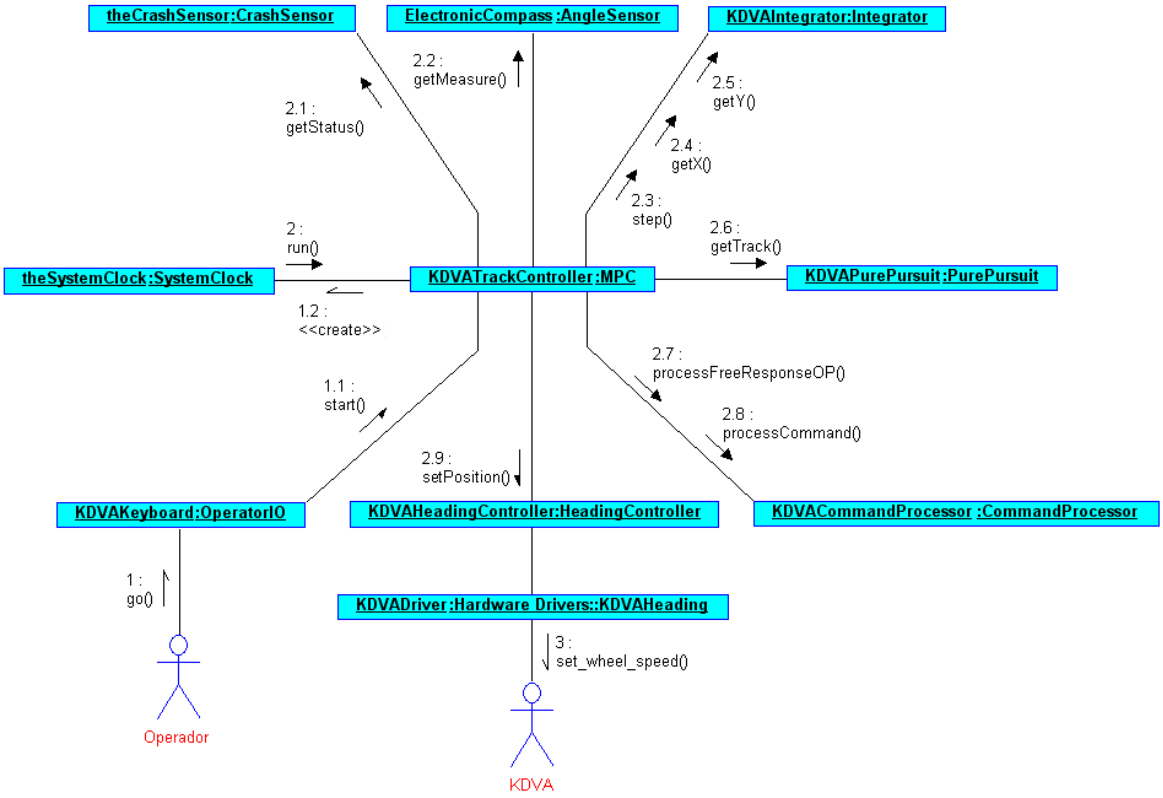

Figura 7.2: Diagrama de colaboração do SCST aplicado ao KDVA.

Como mostra o diagrama, a implementação para o KDVA não faz uso da classe *SpeedController*. Isso se deve ao fato de que, na estrutura de veículos diferenciais, a velocidade e a orientação são controladas através de um sistema único, responsável pelo controle de velocidade das rodas. Assim, a classe *HeadingController*, através da associação com *KDVAHeading*, permite o controle de velocidade e orientação a partir das operações *setPosition* e *setSpeed*. A classe *KDVAHeading* contém o código específico para o acesso ao hardware de acionamento das rodas através da porta paralela

da plataforma X86, bem como o código referente às malhas de controle de velocidade de cada roda.

Após a integração do SCST com o hardware do robô KDVA, foi possível determinar o tempo necessário para que τ_1 se comunique com a bússola eletrônica e com o sistema de controle de velocidade das rodas. Como esta comunicação é realizada por meio da porta paralela, este tempo é determinado pelo tempo das operações de leitura e escrita desta porta, que é da ordem de $2\mu s$. Com isso, estas operações não chegam a influenciar de maneira significativa no tempo de τ_1 apresentado na Tabela 7.1.

7.6.1 Resultados Experimentais

Utilizando a implementação do SCST Completo baseado em modelo cinemático na plataforma X86, foram feitos ensaios experimentais com o veículo KDVA para analisar o desempenho do sistema de controle, por meio de comparação com os resultados simulados. Estes ensaios foram prejudicados em alguns momentos devido a problemas no tacômetro da roda direita, que produziram distúrbios significativos no ângulo de orientação. As figuras 7.3, 7.4 e 7.5 apresentam um ensaio livre de distúrbios mas com trajetória incompleta.

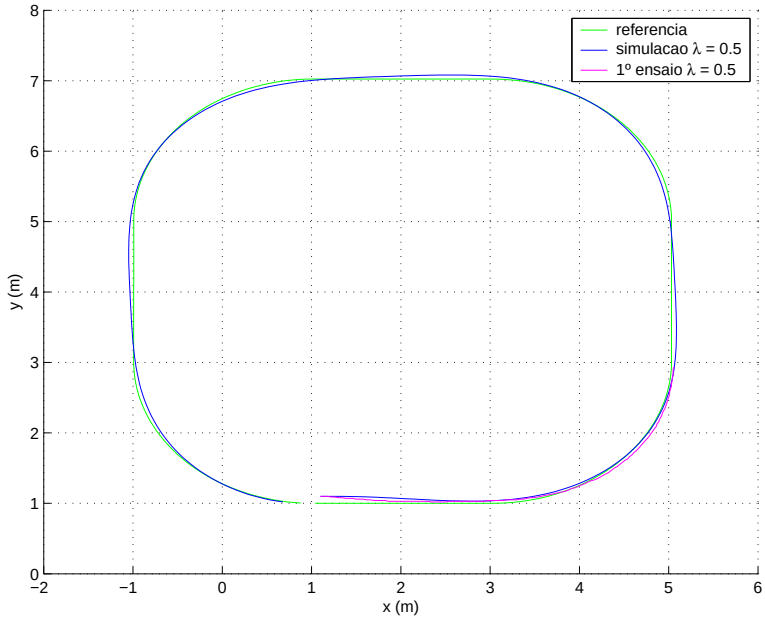

Figura 7.3: Evolução da posição no primeiro ensaio com o KDVA.

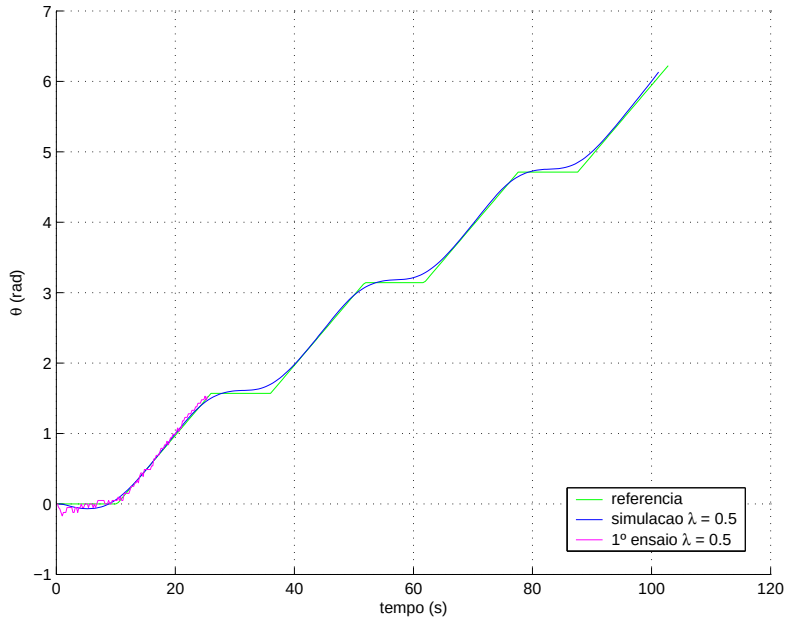

Figura 7.4: Evolução da orientação θ no primeiro ensaio com o KDVA.

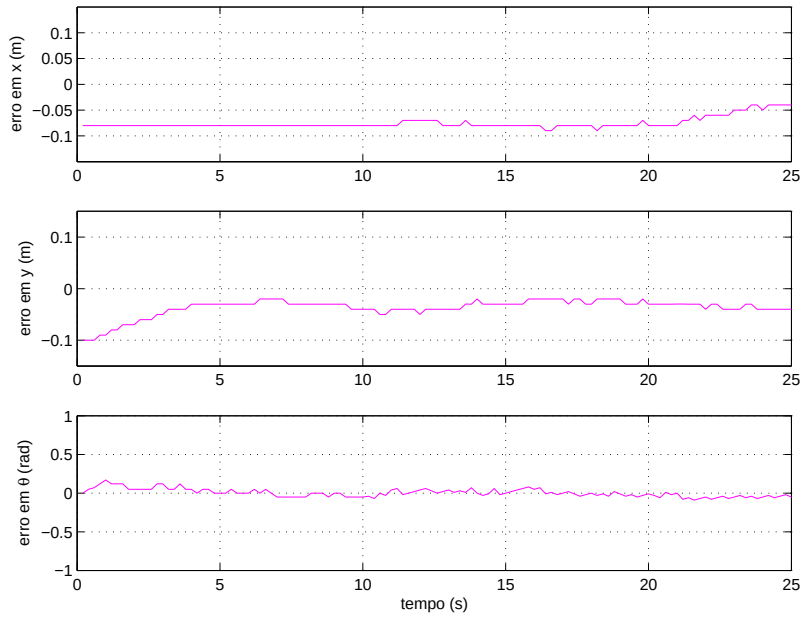

Figura 7.5: Evolução do erro dos estados no primeiro ensaio com o KDVA.

Como se pode observar, embora não tenha sido completada toda a trajetória, a porção percorrida foi suficiente para demonstrar o bom desempenho do SCST e a coerência com o resultado simulado. Um segundo ensaio foi realizando utilizando-se *look-ahead* adaptativo. Porém, desta vez, este ensaio foi muito prejudicado pelo problema no tacômetro, que produziu distúrbios significativos no ângulo de orientação, como mostram as figuras 7.6 e 7.7.

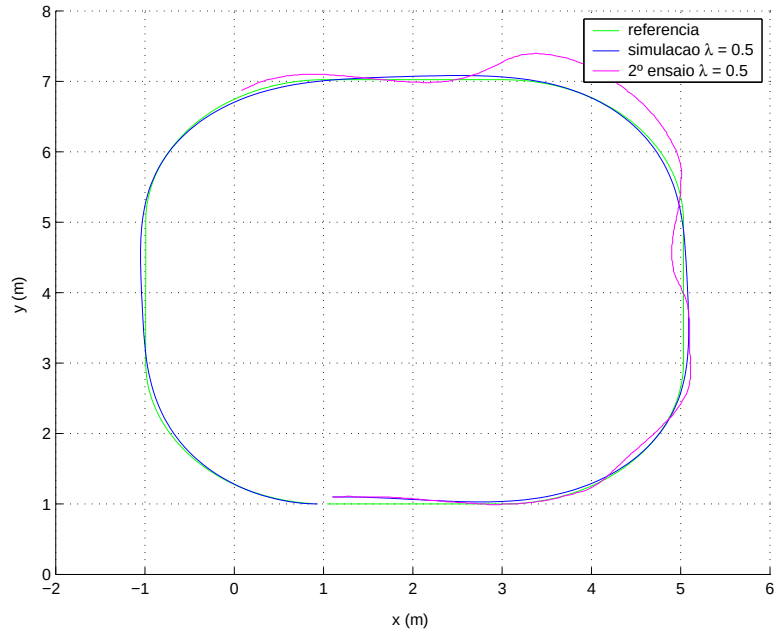

Figura 7.6: Evolução da posição no segundo ensaio com o KDVA.

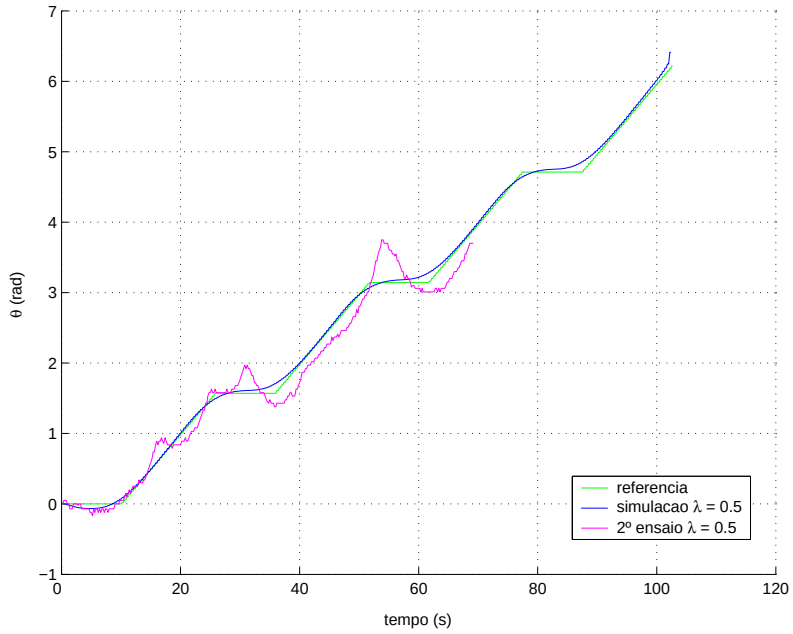

Figura 7.7: Evolução da orientação no segundo ensaio com o KDVA.

Nas figuras pode-se perceber que os distúrbios prejudicaram a estética do percurso e não permitiram uma boa análise do desempenho do sistema utilizando o *look-ahead* adaptativo em condições normais. Porém, estes distúrbios serviram para demonstrar a boa rejeição do controlador auxiliado pela técnica *pure pursuit*, pois mesmo com todas as perturbações, que causaram desvios consideráveis dirigindo o veículo para longe do

percurso, o veículo se encaminhou novamente para a trajetória de referência. A Tabela 7.4 apresenta os parâmetros utilizados nos ensaios:

Parâmetro	**Valor Utilizado**	
	Ensaio 1	Ensaio 2
período de amostragem	$0.2s$	$0.2s$
velocidade linear	$0.2m/s$	$0.2m/s$
N (horizontes)	10	10
Q_θ (peso de θ)	0.9	0.9
Q_y (peso de y_{local})	1	1
R_e (peso do controle)	$312,5$	$312,5$
α (look-ahead)	fixo	adaptativo

Tabela 7.4: Parâmetros de sintonia do controlador nos ensaios com o KDVA.

A partir da análise dos resultados obtidos nos dois ensaios realizados com o veículo KDVA, pode-se concluir que a coerência com os resultados simulados comprova o bom desempenho do sistema de controle com modelo cinemático aplicado a veículos diferenciais. Este bom desempenho pôde ser observado também na rejeição dos distúrbios de medição durante o segundo ensaio.

7.7 Aplicação ao Veículo Mini-Baja

Além dos ensaios realizados com o veículo KDVA, foram realizados também ensaios experimentais utilizando o veículo Mini-Baja apresentado na Figura 7.8, para verificar o desempenho do sistema de controle aplicado a um veículo direcional. Para rodar o SCST embarcado ao veículo Mini-Baja, foi escolhida a plataforma POWERPC pois, como foi visto anteriormente, esta permite a implementação do sistema completo e possui memória suficiente para armazenar os dados do percurso.

Este veículo apresenta as funções de frenagem, aceleração e direção comandadas por sistema *x-by-wire* composto por subsistemas de controle que caracterizam o nível mais baixo da hierarquia da Figura 2.1. Estes sistemas são apresentados em GOMES (2003), KELBER *et al.* (2003) e KELBER *et al.* (2004). No nível da dinâmica, o veículo

Figura 7.8: Veículo Mini-Baja

possui um sistema de controle de velocidade do tipo *Adaptative Cruise Control* (ACC), desenvolvido em GOMES (2003). O veículo pode ser conduzido através de controle tipo manche ou *joystick*, ou remotamente por meio de código DTMF enviado através de telefonia celular, conforme apresentado em KELBER *et al.* (2004). Maiores detalhes sobre o veículo Mini-Baja são apresentados em GOMES (2003).

De modo a realizar os ensaios com o veículo Mini-Baja, a plataforma POWERPC foi integrada ao sistema *x-by-wire* para que o SCST possa atuar sobre o veículo, de forma a conduzi-lo pela trajetória desejada. A integração desta plataforma se deu inicialmente com a definição de como seria implementada, a nível de hardware, a comunicação com o sistema de direção e com a bússola eletrônica. Em seguida, foi desenvolvida uma API para realizar o interfaceamento a nível de software.

O controlador de direção do veículo recebe o comando referente a posição das rodas via rede CANBUS ou sinal analógico. Embora a placa LITE5200 possua duas interfaces CANBUS, o foco deste trabalho não abrange a utilização deste recurso, que foi reservado para aplicações futuras. Portanto, a comunicação com o sistema de direção foi realizada por meio de sinal analógico. Como a placa LITE5200 não possui conversor D/A, foi implementado um conversor de 8 bits que recebe os dados através dos GPIOs

do processador e produz o sinal analógico dentro dos níveis adequados ao controle de direção. No aspecto de TR, a grande vantagem deste método em relação à utilização de comando via rede está na previsibilidade do sistema, pois, apesar das conversões D/A e A/D introduzirem atrasos no envio do comando, estes atrasos são constantes e bem determinados, de modo que podem ser sistematicamente considerados no projeto do sistema, permitindo determinar o tempo de computação da tarefa τ_2.

A bússola eletrônica, apresentada em GOMES *et al.* (2000), fornece a leitura do ângulo de orientação de modo paralelo em padrão TTL, serial em padrão RS232 e sinal analógico. Como a interface RS232 da placa LITE5200 foi utilizada para propósitos de depuração e transmissão dos dados pela tarefa J_5, optou-se por realizar a leitura paralela através dos GPIOs. Esta solução possibilita uma leitura rápida do ângulo de orientação, pois o dado é disponibilizado diretamente nos pinos do processador, e permite a determinação do tempo de computação da tarefa τ_1.

Conforme mencionado no Apêndice B, os GPIOs são compartilhados com os periféricos de comunicação, e para utilizá-los é necessário desabilitar os periféricos. Como a interface RS232 é utilizada para transmissão de dados e pretende-se preservar as duas interfaces CANBUS, foram desabilitadas as interfaces USB e Ethernet. Como os GPIO's correspondentes à interface Ethernet são apenas de saída, estes foram utilizados para comandar a direção por meio do conversor D/A. Já os pinos correspondentes à porta USB foram utilizados para a leitura da bússola eletrônica. A Figura 7.9 apresenta um diagrama que representa o veículo Mini-Baja com o SCST embarcado:

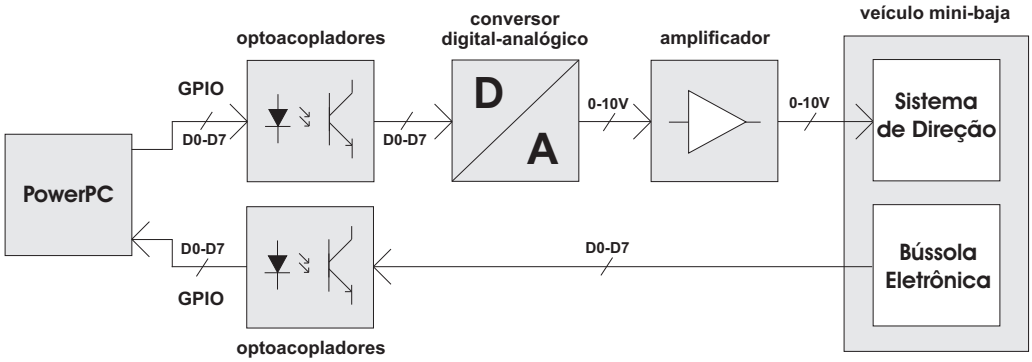

Figura 7.9: Estrututra do SCST embarcado utilizando POWERPC.

Como se pode observar na Figura 7.9, foram utilizados optoacopladores para isolar a

placa LITE5200 da eletrônica do veículo, de modo a evitar sobrecargas ou curto-circuito nos pinos do processador.

Finalizados o projeto e a implementação do hardware de interfaceamento com o sistema *x-by-wire*, foi desenvolvida uma API para o acesso ao hardware representado pela Figura 7.9. Esta API é utilizada pelas classes *MiniBajaHeading* e *MiniBajaSpeed*, implementadas a partir da derivação das classes *HeadingDriver* e *SpeedDriver*, respectivamente, e que operam como drivers para a utilização do sistema *x-by-wire* do veículo. A Figura 7.10 apresenta o diagrama de colaboração com a estrutura lógica implementada para os ensaios com o veículo Mini-Baja.

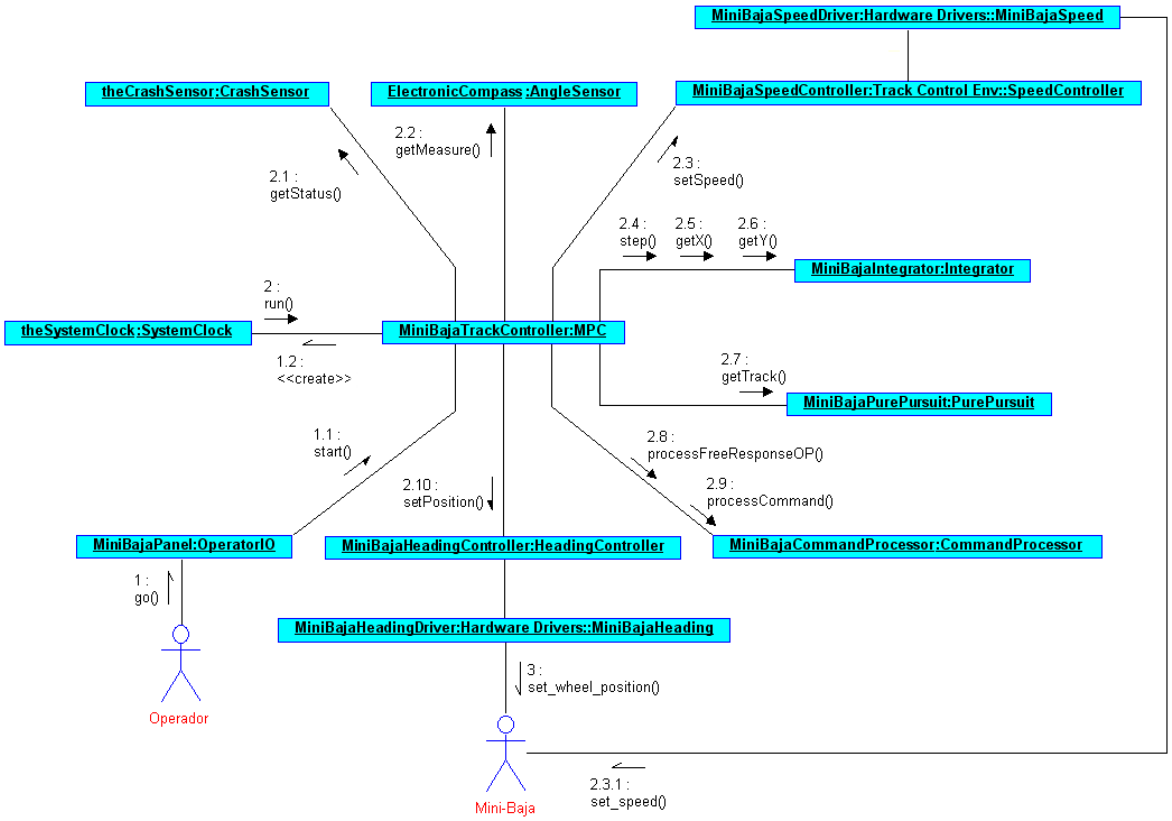

Figura 7.10: Diagrama de colaboração do SCST aplicado ao Mini-Baja.

Como mostra o diagrama, ao contrário do controle de robôs diferenciais como o caso do KDVA, aqui se faz necessário o uso da classe *SpeedController*, pois em robôs direcionais os sistemas de controle de direção e velocidade são independentes. Assim, através da associação com as classes *MiniBajaHeading* e *MiniBajaSpeed*, as classes *HeadingController* e *SpeedController* permitem ao SCST comandar o veículo Mini-Baja sem o conhecimento dos detalhes de hardware do sistema *x-by-wire*.

Quanto à influência sobre τ_1 dos tempos de comunicação com a bússola eletrônica, controle de velocidade e controle de direção, tem-se que esta influência é insignificante, uma vez que estes tempos de comunicação consistem apenas no tempo de escrita ou leitura do registrador correspondente ao barramento desejado e o tempo de propagação dos optoacopladores.

7.7.1 Resultados Experimentais

Após concluída a integração da plataforma POWERPC aos subsistemas do veículo, foram realizados os ensaios experimentais para verificar o desempenho do sistema. O primeiro ensaio foi realizado utilizando-se o *look-ahead* fixo, sendo que o resultado obtido e a comparação deste com o resultado simulado podem ser vistos nas figuras 7.11, 7.12 e 7.13.

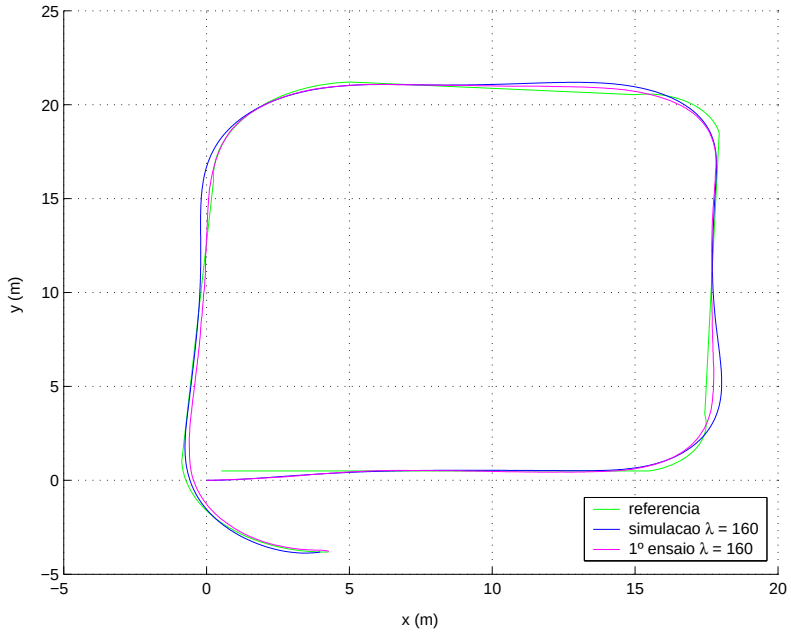

Figura 7.11: Evolução da posição no ensaio com o Mini-Baja utilizando trajetória retangular.

Como se pode observar, o resultado experimental é coerente com o resultado simulado, validando mais uma vez as análises teóricas realizadas no Capítulo 4. Os resultados também deixam claro que a técnica de *look-ahead* fixo implica diferenças

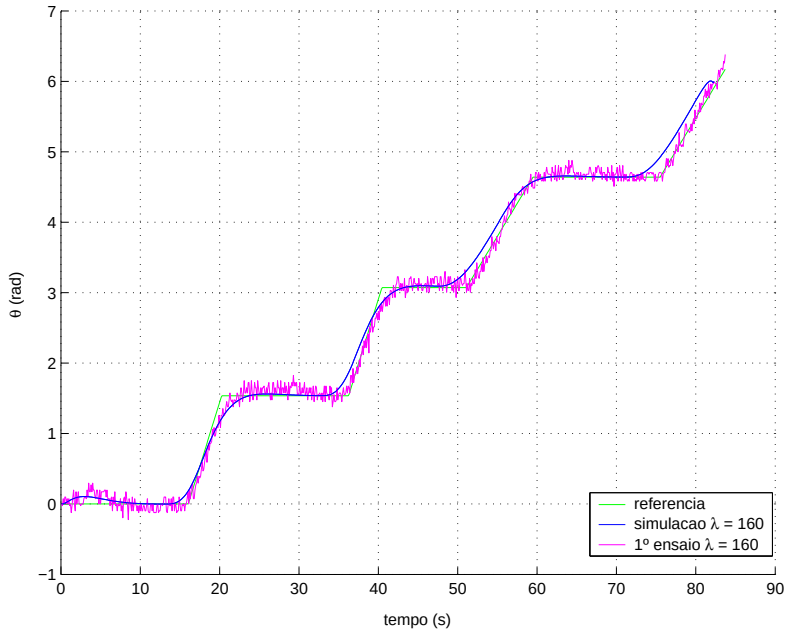

Figura 7.12: Evolução da orientação no ensaio com o Mini-Baja utilizando trajetória retangular.

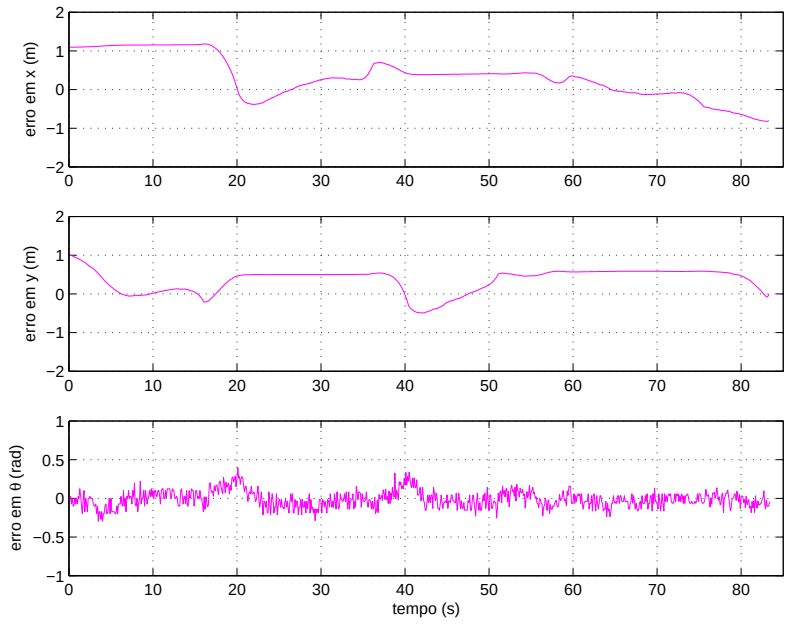

Figura 7.13: Evolução do erro do estados no ensaio com o Mini-Baja utilizando trajetória retangular.

significativas entre a trajetória desejada e a percorrida em alguns pontos, o que fica claro na Figura 7.13. Isso se deve ao fato de que o *look-ahead* neste caso, é ajustado em função da velocidade do veículo e da distância inicial deste em relação ao início

da trajetória. Assim, embora o *look-ahead* cumpra bem a função inicial de trazer o veículo até a trajetória, ele pode não ser adequado para guiar o veículo durante partes específicas do percurso, como, por exemplo, durante as curvas da trajetória, como mostra a Figura 7.11. Na realidade, isso também é atribuído ao fato de a trajetória setada para este primeiro ensaio não ser a ideal, pois para se adequar ao espaço físico disponível, tiveram que ser usadas curvas muito fechadas, próximas ao limite de curvatura do veículo. O desempenho no percurso desta trajetória pode ser melhorado através da utilização da técnica de *look-ahead* adaptativo ou ainda através da diminuição da ponderação do esforço de controle e aumento do horizonte de predição, embora estas alterações impliquem na imposição de ângulos δ_D próximos ao limite físico do veículo.

Utilizando-se a técnica de *look-ahead* adaptativo, foram realizados dois outros ensaios utilizando-se diferentes sintonias do controlador. Os resultados são apresentados nas figuras 7.14, 7.15 e 7.16.

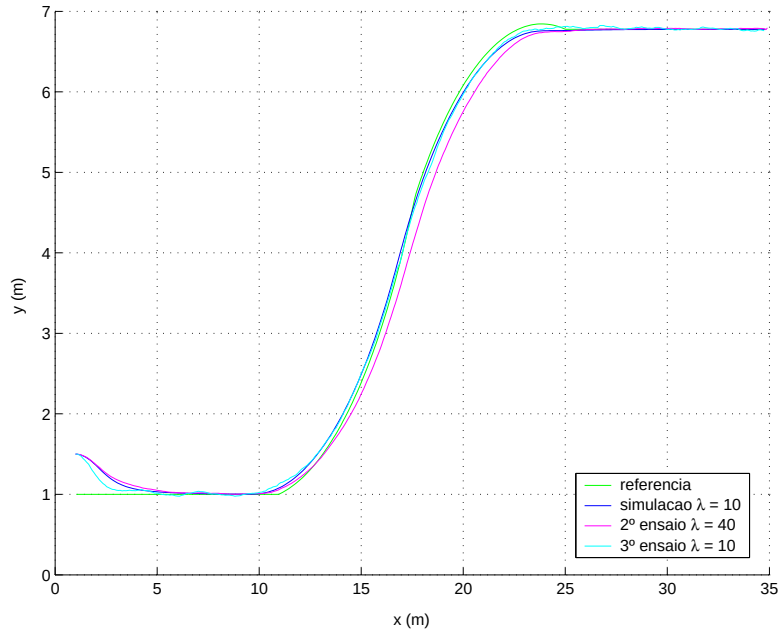

Figura 7.14: Evolução da posição nos ensaios com o Mini-Baja utilizando trajetória em S.

Os resultados com *look-ahead* adaptativo comprovam que esta técnica proporciona uma melhora no desempenho do sistema, uma vez que o veículo se posiciona melhor durante as curvas. Isso fica claro na Figuras 7.16, onde percebe-se uma redução no

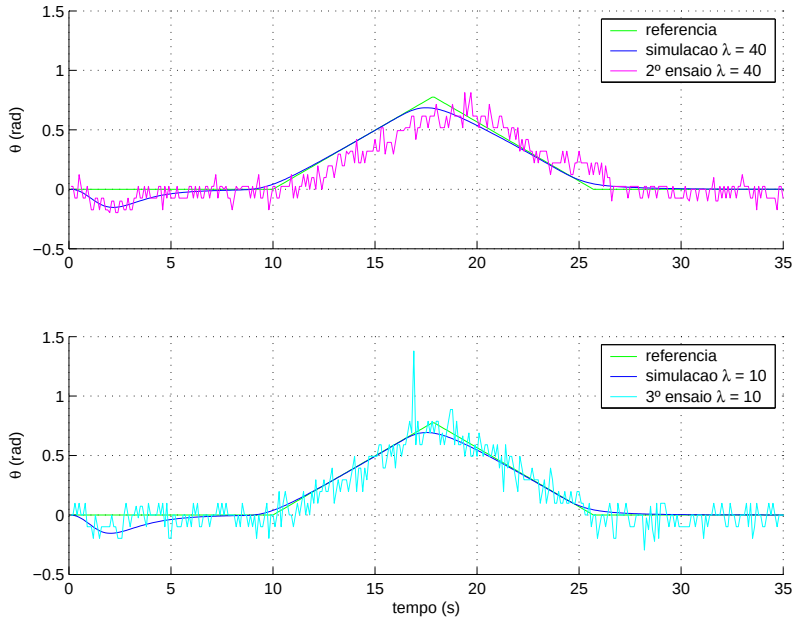

Figura 7.15: Evolução da orientação nos ensaios com o Mini-Baja utilizando trajetória em S.

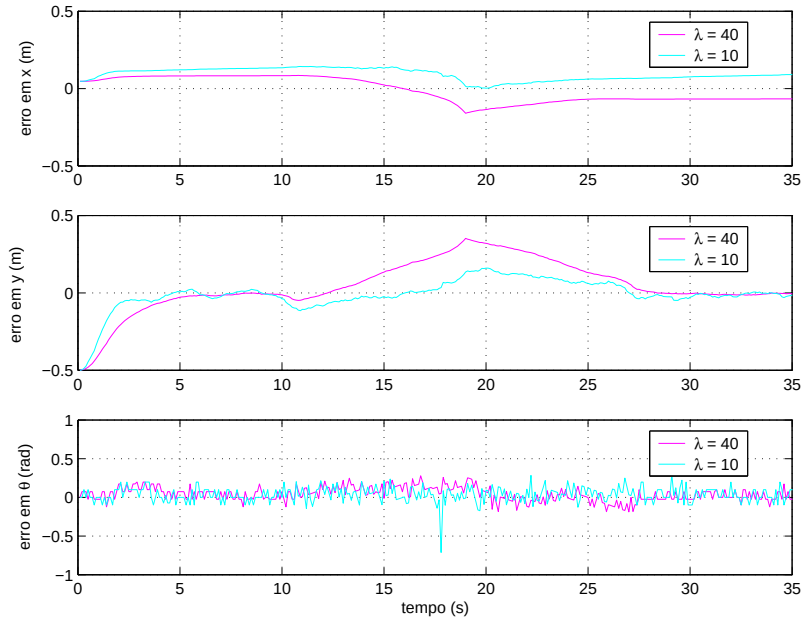

Figura 7.16: Evolução do erro dos estados nos ensaios com o Mini-Baja utilizando trajetória em S.

erro dos estados em relação ao resultado da Figura 7.13. Porém, esta comparação também deixa claro que ocorre uma pequena tendência oscilatória, devido à redução significativa do *look-ahead* à medida em que o veículo se aproxima da trajetória. Isso implica a necessidade de se limitar o valor mínimo para evitar a perda de estabilidade.

Na Figura 7.16, também se pode observar que a diminuição do fator λ, assim como ocorre com o valor do *look-ahead*, reduz o erro no seguimento da trajetória às custas de um comportamento oscilatório. Os parâmetros de sintonia do controlador utilizados nos ensaios com o Mini-Baja são apresentados na Tabela 7.5.

Parâmetro	**Valor Utilizado**		
	Ensaio 1	Ensaio 2	Ensaio 3
período de amostragem	$0.05s$	$0.05s$	$0.05s$
velocidade linear	$1m/s$	$1m/s$	$1m/s$
N (horizontes)	30	30	30
Q_θ (peso de θ)	0.85	1	1
Q_y (peso de y_{local})	1	1	1
R_e (peso do controle)	90630	22658	5664
α (look-ahead)	fixo	fixo	adaptativo

Tabela 7.5: Parâmetros de sintonia do controlador nos ensaios com o Mini-Baja.

Analisando-se os dados da Tabela 7.5, pode-se observar que o período de amostragem de $50ms$ escolhido para controlar o veículo com segurança a uma velocidade de $1m/s$, é menor que o tempo de $70,566ms$, necessário para execução da tarefa J_2, conforme a Tabela 7.3. Assim, este cenário já caracteriza um caso de necessidade de algoritmo de escalonamento preemptivo para garantir o cumprimento de todas as restrições temporais se houver necessidade de ajuste de modo de operação durante o percurso.

7.8 Análise da Degradação de Desempenho

Diante da possibilidade de implementação do SCST em diferentes plataformas embarcadas e em robôs também com características diferenciadas, alguns fatores relacionados às características específicas da arquitetura de cada aplicação podem influenciar no tempo necessário para o cálculo da ação de controle. Como foi analisado durante a modelagem do sistema, a ação de controle é calculada periodicamente sendo resultado da execução da tarefa τ_1. Pode-se dizer que o início do cálculo da ação de controle

se dá a partir da leitura do ângulo de orientação através da bússola eletrônica e termina com a aplicação efetiva do comando ao sistema de direção do veículo. Em alguns cenários de aplicação do SCST, o atraso na execução das operações pode variar ao longo dos períodos de amostragem devido a diversos fatores, como influência do algoritmo de escalonamento do SOTR e atrasos na comunicação entre subsistemas nos diferentes níveis representados pela Figura 2.1. Na prática, esta variação no tempo de execução das ações de τ_1 acaba ocasionando uma oscilação no intervalo Δt entre as aplicações da ação de controle, como mostra a Figura 7.17.

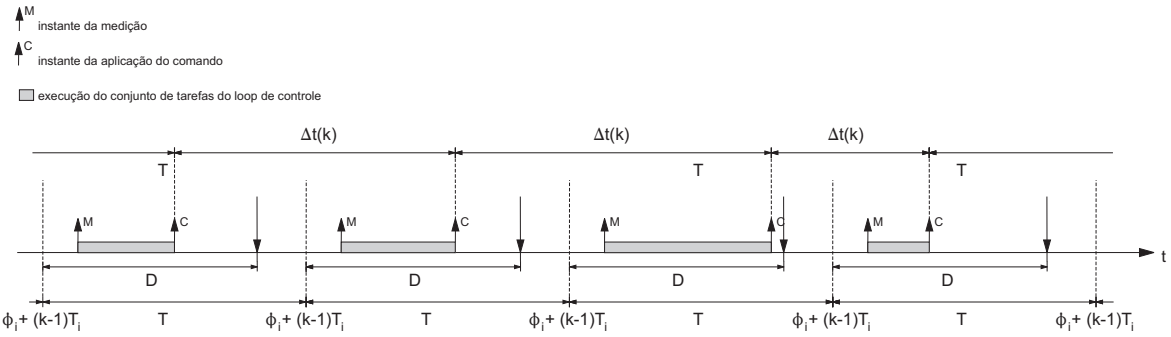

Figura 7.17: Atraso na execução das tarefas.

Para analisar a influência desta variação na degradação do desempenho do sistema, foram realizadas simulações do SCST baseado em modelo cinemático sujeito a oscilações randômicas de 30% no intervalo de tempo entre as aplicações da ação de controle. As simulações realizadas utilizando-se o modelo de veículo diferencial podem ser vistas na Figura 7.18, enquanto que as simulações realizadas com modelo direcional são apresentadas nas figuras 7.19 e 7.20.

Os resultados das simulações mostram que o efeito da variação do atraso na execução do cálculo da ação de controle influencia no desempenho do sistema, pois o veículo acaba realizando o percurso a uma certa distância dos pontos de referência. Como conseqüência da variação do período entre a aplicação das ações de controle, tem-se um aumento do erro de integração, principalmente em percursos curvilíneos, devido ao fato de o integrador considerar constante os intervalos entre os comandos aplicados ao veículo. Desta forma, é desejável que a aplicação tempo-real do SCST procure minimizar estas variações de modo a garantir maior fidelidade do percurso em relação à trajetória pré-determinada.

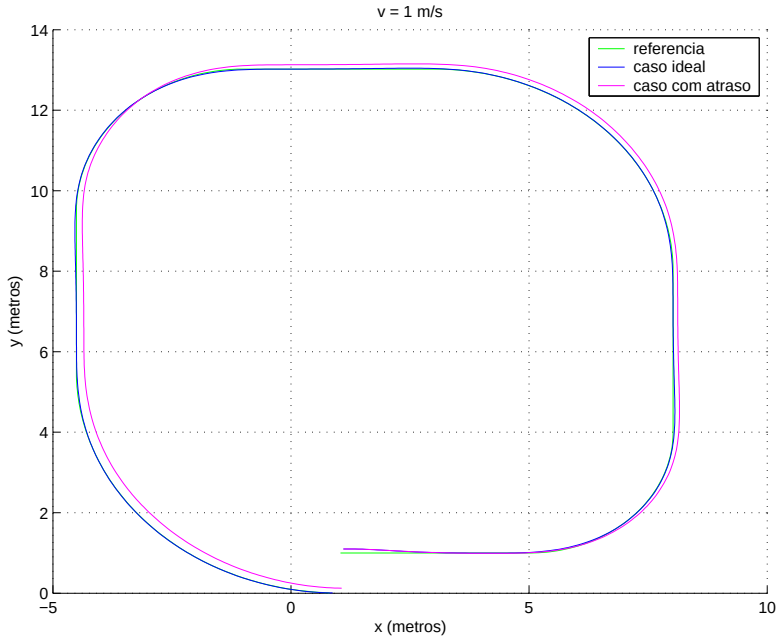

Figura 7.18: Simulação do atraso na execução das tarefas com veículo diferencial.

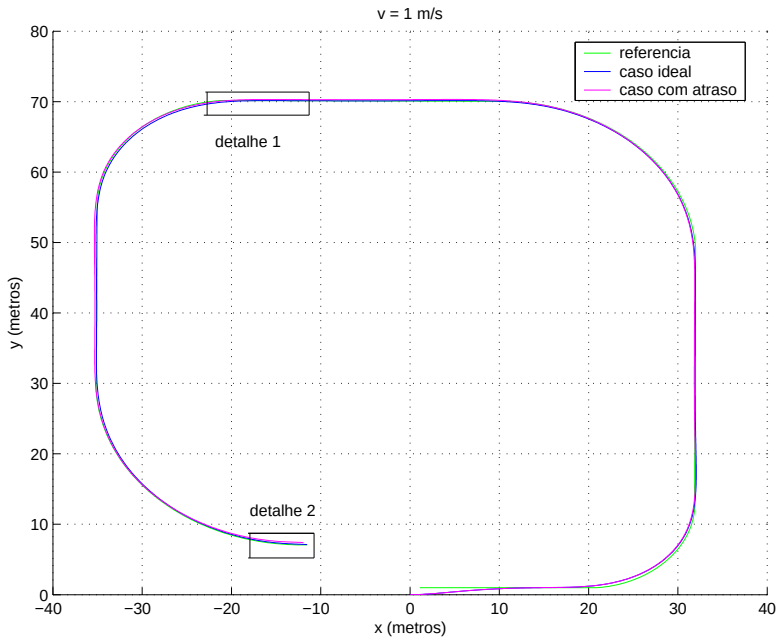

Figura 7.19: Simulação do atraso na execução das tarefas com veículo direcional.

7.8.1 Conclusões

A partir dos resultados obtidos nos ensaios, pôde-se comprovar a eficácia da estratégia utilizada para o controle de seguimento de trajetória. Os resultados demonstraram o bom desempenho da técnica de controle preditivo utilizando coordenadas locais e

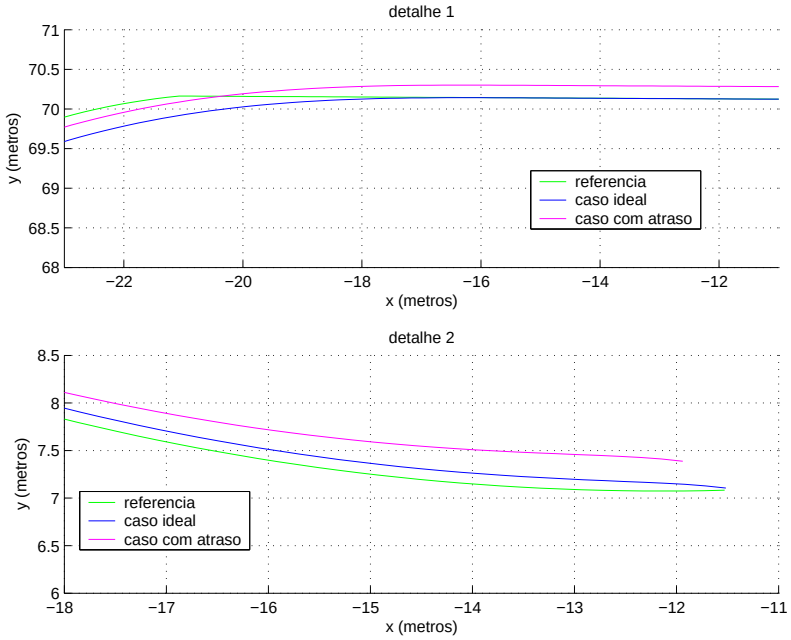

Figura 7.20: Detalhes da Figura 7.19.

geração de trajetória de aproximação baseada no algoritmo *pure pursuit*. A utilização do sistema de coordenadas locais apresenta um boa relação entre custo computacional e desempenho, apresentando tempos de execução satisfatórios em todas as plataformas testadas. A estratégia de trajetória de aproximação baseada no algoritmo *pure pursuit* mostrou-se um fator importante a nível de sintonia, pois influencia diretamente na capacidade de curvatura do veículo e na estabilidade do sistema de controle. A utilização de *look-ahead* adaptativo permite ao veículo percorrer uma trajetória mais próxima da referência, mas implica a escolha cautelosa do valor mínimo para evitar instabilidades.

Com relação às questões de tempo-real, tem-se que a determinação dos tempos de execução foi útil para avaliar a resposta de cada plataforma frente à carga computacional imposta pelo algoritmo de controle, o que ajudou a determinar a plataforma mais adequada a cada tipo de aplicação, e também forneceu dados que são úteis para o ajuste dos mecanismos de garantia necessários a certas aplicações. Através de simulação computacional, foi também analisada a influência de atrasos na execução do cálculo da ação de controle no desempenho do sistema, em que se verificou que estes atrasos provocam um aumento no erro de integração, ocasionando em conseqüência, erros no seguimento da trajetória. Assim, é desejável que a aplicação em tempo-real procure também minimizar estes atrasos de modo a se obter maior precisão no percurso

da trajetória.

Quanto à modelagem OO, verificou-se que sua utilização foi benéfica no sentido de organizar a implementação do código com estrutura lógica favorável a implementação em tempo-real. Além disso, foi possível tornar o código modular e de fácil adaptação, como observado nas classes de acesso ao hardware específico de cada robô e nos métodos de integração e geração de trajetória de aproximação, que podem ser modificadas visando a utilização de diferentes técnicas. A modelagem realizada também promove a reutilização de software, uma vez que a classe *MPC* pode ser utilizada em diferentes aplicações de controle e plataformas.

Em relação à implementação nas plataformas DSP e POWERPC, tem-se que a limitação imposta pela necessidade de utilização de linguagem C impediu que fosse tirado proveito de todos os benefícios do projeto e desenvolvimento OO, benefícios estes que poderiam ser obtidos através de código escrito também em linguagem OO, como o C++.

Capítulo 8

Conclusões Finais

Este trabalho se dedicou ao estudo, desenvolvimento e implementação de um sistema de controle tempo-real para tratar o problema do seguimento de trajetória de robôs móveis diferenciais e direcionais. O sistema desenvolvido utiliza técnica de controle preditivo para controlar os comportamentos cinemáticos e dinâmicos de robôs móveis, podendo ser utilizado tanto em aplicações simples de robótica móvel quanto em aplicação mais severas, caracterizadas por velocidades elevadas e transporte de cargas expressivas.

Os objetivos pretendidos com a utilização de algoritmo de controle preditivo para o sistema de controle, motivada pelo fato de esta técnica apresentar características favoráveis ao tratamento do problema de seguimento de trajetória, foram atingidos. O CPBM apresentou resultados satisfatórios que, em parte, são justificados pela capacidade de utilizar referência futura, tirando proveito da existência de trajetória pré-determinada na maioria das aplicações de veículos autônomos. Isso permite a reação antecipada do veículo frente às curvas ao longo do percurso.

A escolha do algoritmo GPC é vantajosa, pois a utilização de função de transferência para representar o modelo do processo permite a identificação do modelo a partir de ensaios experimentais e métodos de identificação, além de ser favorável a aplicações de natureza adaptativa. O algoritmo GPC, aplicado ao controle do comportamento cinemático de veículos diferenciais e direcionais utilizando modelo linearizado, proporcionou bom compromisso entre custo computacional e desempenho. A análise dos tempos de execução do código nas plataformas analisadas mostrou que o GPC apresenta custo computacional aceitável para a execução em tempo-real na maioria dos sistemas computacionais destinados a sistemas embarcados. O GPC, aplicado ao controle do

comportamento dinâmico com modelo linearizado, também apresenta vantagens, pois, além do menor custo computacional, a utilização de função de transferência dispensa o uso de estruturas dinâmicas, como observadores de estados, que seriam necessários em outras técnicas, como o CPBM no espaço de estados.

A utilização de método de geração de trajetórias de aproximação baseado no algoritmo *pure pursuit*, demonstrou-se muito importante e eficaz para garantir as condições de validade do modelo cinemático linearizado, além de contribuir para a condução do veículo de forma suave e estável. O ajuste do parâmetro *look-ahead* se demonstrou um fator importante na sintonia do sistema de controle, influenciando de forma significativa na qualidade do seguimento do percurso e na estabilidade do sistema. A partir da constatação de que o ajuste adequado do *look-ahead* para aproximar o veículo de pontos distantes até a trajetória passa a não ser mais ideal no momento em que este está sobre a trajetória e vice-versa, foi verificada a necessidade de método de adaptação on-line de forma a manter o correto ajuste do *look-ahead* ao longo de todo o percurso.

De modo a possibilitar a implementação do controle em diferentes plataformas embarcadas, foram modelados dois modos de implementação do sistema, o modo simplificado e o modo completo. O modo simplificado é destinado a aplicações em que as condições de operação são constantes ou sofrem poucas variações, de modo que os parâmetros de configuração possam ser definidos off-line, resultando em menor custo computacional e uso de memória, o que permite a implementação em hardware mais restrito. Já o modo completo é destinado a aplicações mais complexas e de alto desempenho, em que as condições de operação podem variar significativamente, o que remete à necessidade de ajuste on-line de parâmetros de configuração, implicando maior custo computacional e uso de memória.

Como este é um sistema tempo-real, foram identificadas as restrições temporais presentes em cada um dos modos de implementação do sistema. Verificou-se a possibilidade de uso de escalonamento off-line nos modos de implementação completo e simplificado, embora o modo completo apresente uma maior otimização com escalonamento on-line. Também no modo completo, ocorre a necessidade de utilização de mecanismo de controle de acesso para a área de memória que armazena o modelo do processo e a lei de controle, de modo a evitar que a execução concorrente da definição da

lei de controle e do cálculo da ação de controle realizem acessos simultâneos de leitura e escrita. O mecanismo de controle de acesso aos recursos compartilhados não deve permitir preempção durante tais acessos, de modo a impedir que tarefas de escrita sejam interrompidas e, conseqüentemente, dados inconsistentes sejam deixados na memória. Ainda sobre as características do algoritmo de escalonamento para o sistema completo, tem-se que este deve ser preemptivo de modo a poder distribuir a execução do ajuste do modo de operação ao longo de alguns períodos de amostragem, pois esta tarefa em alguns casos apresenta o tempo de execução muito elevado. Em aplicações em que há a necessidade de correção de erro de integração, ambos os modos de implementação requerem mecanismo de controle de acesso à memória de integração das coordenadas, uma vez que a correção do erro e o cálculo da ação de controle são tarefas executadas concorrentemente e que realizam, respectivamente, escrita e leitura deste recurso.

Nas etapas de modelagem e desenvolvimento do sistema de controle, a utilização do paradigma de orientação a objetos permitiu a obtenção de código fonte modular, de fácil adaptação e manutenção, cuja estrutura lógica estimula a reutilização além de ser favorável ao emprego de tecnologias específicas de implementação em tempo-real. A pouca necessidade de adaptação do código durante a implementação em cada plataforma testada também demonstrou a boa portabilidade deste. A utilização de linguagem UML associada ao perfil UML-TR permitiu a modelagem de forma clara da estrutura de classes e seu relacionamentos, bem como a representação adequada dos principais requisitos de tempo-real. Como vantagens concretas da implementação OO, pode-se citar a facilidade de adaptação do sistema a técnicas diferenciadas de integração e geração de trajetória de aproximação e a possibilidade de reutilização do pacote *GPC* em diversas aplicações de controle de processos. Com isso, atingiram-se também os objetivos da utilização do paradigma de orientação a objetos e da análise das restrições temporais.

Na etapa de implementação, os resultados experimentais obtidos com a aplicação do sistema de controle a veículos diferenciais e direcionais comprovaram a boa performance da técnica proposta para tratar o problema do seguimento de trajetória. Os ensaios com *look-ahead* fixo e adaptativo também acompanharam os resultados das simulações, confirmando a necessidade de adaptação periódica para a obtenção de boa performance

tanto na aproximação da trajetória quanto na permanência sobre esta. A análise, com base em simulação, da implementação em arquitetura distribuída mostrou que atrasos significativos na execução de tarefas, mesmo sem perda de *deadlines*, implicam degradação de performance no seguimento de trajetória, uma vez que a variação no intervalo de tempo entre a aplicação dos comandos do controlador acentua o erro de integração.

Por fim, a partir das análises, simulações e resultados experimentais realizados com SCST proposto, se conclui que foi atingido o objetivo maior deste trabalho, que consiste no desenvolvimento de sistema tempo-real embarcado baseado em técnica de controle preditivo para tratar o problema de seguimento de trajetória de veículos diferenciais e direcionais. Assim, este trabalho contribui para a área de robótica móvel através da análise detalhada de aspectos relativos ao sistema de controle proposto, à estruturação das etapas de projeto e implementação, e aos resultados experimentais.

Como sugestões para trabalhos futuros na área de controle, propõem-se a realização de ensaios experimentais com a estrutura de controle em cascata para controlar também o comportamento dinâmico, a adaptação do algoritmo para consideração de restrições no cálculo da lei de controle e também a implementação e teste de técnicas diferenciadas de geração de trajetória, a exemplo de *vector pursuit* e $G^3 - spline$, de modo a realizar uma correlação entre desempenho e custo computacional. Na área de tempo-real, é sugerida a implementação do código utilizando as opções de sistemas operacionais de tempo-real disponíveis para as plataformas DSP e POWERPC, bem como a utilização de mecanismos de tratamento de exceções e tolerância a falhas. Assim, verifica-se a grande possibilidade de extensão do projeto de pesquisa, e em áreas importantes que podem contribuir significativamente para a melhoria do sistema de controle desenvolvido.

Apêndice A

Pacotes com as Classes do Sistema

As classes desenvolvidas no projeto do SCST que apresentam comportamentos semelhantes foram organizadas em pacotes com funções específicas, de modo a promover a reutilização de software. Estes pacotes são apresentados a seguir.

A.1 GPC

Pacote com as classes contendo o algoritmo *Generalized Predictive Controler* (GPC) (CLARKE *et al.*, 1987), apresentado no Capítulo 4. O Pacote e suas classes detalhadas são apresentadas na Figura A.1.

A classe *GPC* oferece o algoritmo GPC completo, sendo destinada à aplicações sem a necessidade de execução concorrente.

A classe *ControlLawProcessor* oferece operações específicas para o cálculo da lei de controle GPC, dada por $(\mathbf{G}' \cdot \mathbf{Q} \cdot \mathbf{G} + \mathbf{R})^{-1} \cdot \mathbf{G}' \cdot \mathbf{Q}$ da expressão (4.36).

A classe *ControlLawProcessor* é destinada ao cálculo da predição ótima, dada pela expressão (4.29).

A classe *MPC*, através de relação de agregação com as classes *ControlLawProcessor* e *CommandProcessor*, oferece as mesmas operações da classe *GPC*. Porém, *MPC* apresenta autonomia lógica entre o cálculo da lei de controle e da ação de controle, exatamente pela agregação de instâncias de *ControlLawProcessor* e *CommandProcessor*, o que a torna adequada à execução concorrente.

Figura A.1: Pacote *GPC*.

A.2 Track Control Env

O ambiente necessário ao controle do comportamento cinemático com modelo linearizado em coordenadas locais motivou a criação do pacote *Track Control Env*, apresentado na Figura A.2, que contém as classes *PurePursuit* e *Integrator*. A primeira oferece operações para a geração de trajetória suave com base no algoritmo *pure pursuit* (AMIDI, 1990). A segunda oferece método de integração numérica para determinar a evolução das coordenadas cartesianas do veículo no plano de deslocamento. Estas classes podem ser utilizadas junto a outras técnicas para tratar o problema de seguimento de trajetória.

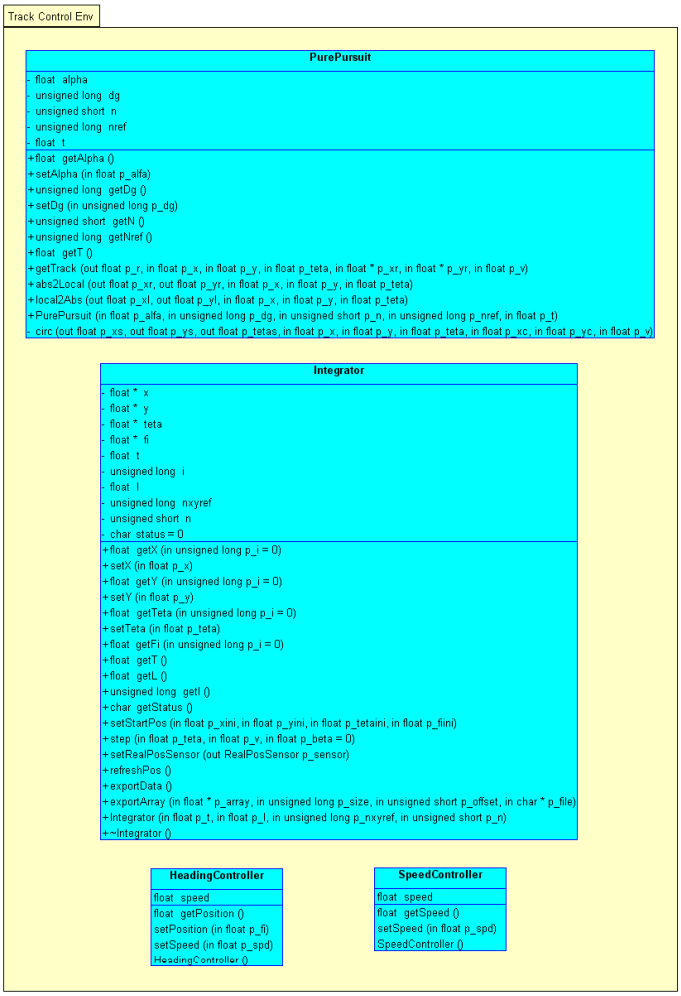

Figura A.2: Pacote *Track Control Env*.

A.3 Track Control Sensores

As classes que modelam os sensores foram agrupadas no pacote *Track Control Sensors* que pode ser visto na Figura A.3.

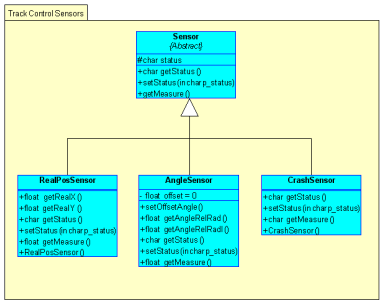

Figura A.3: Pacote *Track Control Sensores*.

A.4 Hardware Drivers

As classes que modelam os drivers de acesso a hardware do processo a ser controlado, e em específico aos veículos KDVA e Mini-Baja, foram agrupadas no pacote *Hardware Drivers* que pode ser visto na Figura A.4.

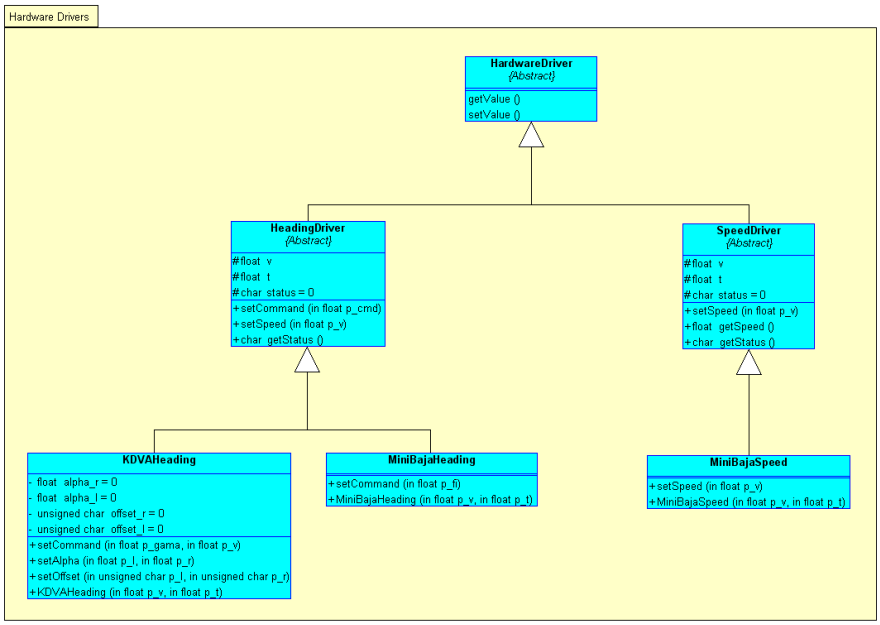

Figura A.4: Pacote *Hardware Drivers*.

Apêndice B

Características das Plataformas Utilizadas

B.1 Caracterísitcas do kit DSP 56F8001

Para realizar a implementação do SCST em plataforma DSP, foi utilizado um kit com o DSP 56F801 da Motorola, que apresenta uma arquitetura de 16 bits e opera a uma freqüência de clock de 60MHz. Este DSP é específico para aplicações de controle de motores, sendo que algumas das suas principais características são:

- Unidade Lógica e Aritmética (ULA) de 16 bits, dois acumuladores de 36 bits e todos os registradores acessíveis como origem ou destino de operações aritméticas e acumulador de 36 bits;
- 15 modos diferentes de endereçamento, proporcionando código de tamanho reduzido e programação facilitada;
- Múltiplos barramentos de endereçamento e dados, possibilitando acessos e movimentações simultâneas;
- Dois níveis de interrupções aninháveis, proporcionando grande flexibilidade na definição de prioridade de interrupções;
- Mecanismo de controle de loop por hardware, possibilitando execução rápida de código iterativo, e proporcionando transparência em aplicações de TR;
- Unidade manipuladora de bits que permite um eficiente controle de código e programação de periféricos;

- Suporta memórias de diferentes velocidades, de modo a atender compromissos de custo/performance, através de estados de espera de memória (*memory wait states*) programáveis por software;

- Múltiplos modos de baixo consumo, permitindo redução significativa do consumo de energia.

Estas características são proporcionadas pela arquitetura específica do núcleo do DSP 56F801, que é representada pelo diagrama de blocos da Figura B.1, onde observa-se a presença de múltiplos barramentos de endereço e dados. Para a memória de programa, existem dois barramentos exclusivos para endereçamento de dados (PAB e PDB). A memória interna é acessada por meio dos barramentos de dados XAB1 e XAB2, sendo que XAB1 também realiza acessos à memória externa. A transferência de dados para a ULA e o restante do núcleo do processador ocorre pelo barramento de dados CGDB. Já os dados originados ou destinados a periféricos utilizam o PGDB. Esta estrutura de barramentos suporta transferências de registrador para registrador, registrador para memória, memória para registrador e pode realizar até três transferências de palavras de 16 bits em um único ciclo de instrução.

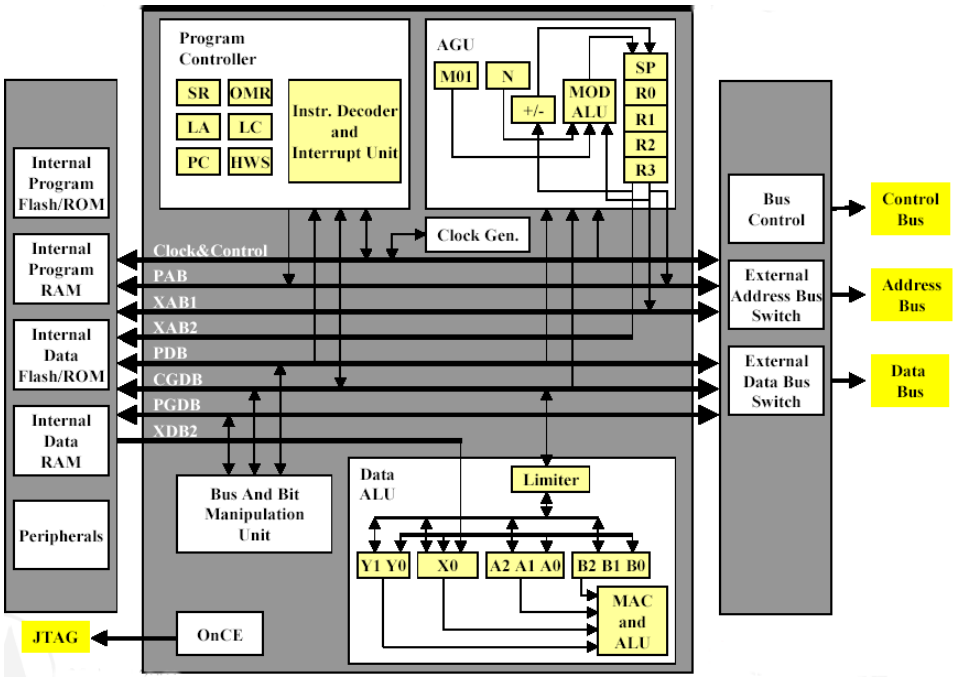

Figura B.1: Arquitetura do DSP 56F8001.

Além destas características do núcleo do processador, o DSP 56F8001 ainda apresenta os seguintes periféricos:

- Modulador por largura de pulso ou *Pulse Width Modulator* (PWM);
- Conversor analógico-digital de oito canais e resolução de 12 bits, com opções de interrupções programáveis ativadas por nível de tensão, por cruzamento de nível zero e por início e fim de conversão;
- *Quad-Timer* de três canais com interrupções programáveis ativadas por início, fim e reinício de contagem, *overflow* e valores específicos de contagem.

Estes periféricos são disponibilizados de forma otimizada para aplicações específicas de controle de motores elétricos, sendo comandados pelo processador por meio pinos de entrada e saída de uso geral (*General Purpouse I/O* GPIOs). Quando os periféricos não são necessários, eles podem ser desabilitados via software de modo a permitir a utilização dos GPIOs para outros propósitos.

B.2 Caracterísitcas do kit LITE5200

A implementação do SCST na plataforma POWERPC foi realizada utilizando-se o kit LITE5200, da Freescale, que utiliza o POWERPC MPC5200 da Motorola, um processador de 32 bits que opera a uma freqüência de clock de 400MHz. O núcleo do processador possui dupla precisão na Unidade de Ponto Flutuante (*Float Point Unit*, FPU), o que é crucial para processamento de dados em aplicações de reconhecimento de voz, processamento de vídeo, GPS, e outras aplicações de intenso uso matemático. O kit LITE5200 ainda possui 16MBytes de memória flash, 64MBytes de memória RAM e os seguintes periféricos de comunicação:

- Uma porta USB 1.1 compatível apenas em Master;
- Uma interface Ethernet 10/100Mbps BaseT;
- Uma porta serial RS232;

- Duas portas CANBUS 2.0 A/B de alta velocidade, frames padrão e extendido e taxa de bits programável de até 1Mbps;
- Um barramento PCI;
- Um barramanto IDE;
- Uma porta J1850, baseada em protocolo desenvolvido pela *Society of Automotive Engineers* (SAE);
- Uma porta para comunicação entre integrados I^2C;
- Controlador serial de periféricos PSC1;
- Interface de comunicação serial SPI;
- Uma barramento para comunicação de discos rígidos ATA;
- Porta de depuração COP/JTAG;
- Sistema de áudio "*direct sound*" AC97.

Muitas destas interfaces compartilham pinos de entrada e saída de uso geral (GPIOs), de modo que quando um destes periféricos é desabilitado, os respectivos pinos ficam disponíveis para utilização. A Figura B.2 apresenta um diagrama com a arquitetuta de hardware do processador MPC5200 e a Figura B.3 apresenta a estrutura de periféricos do kit LITE5200.

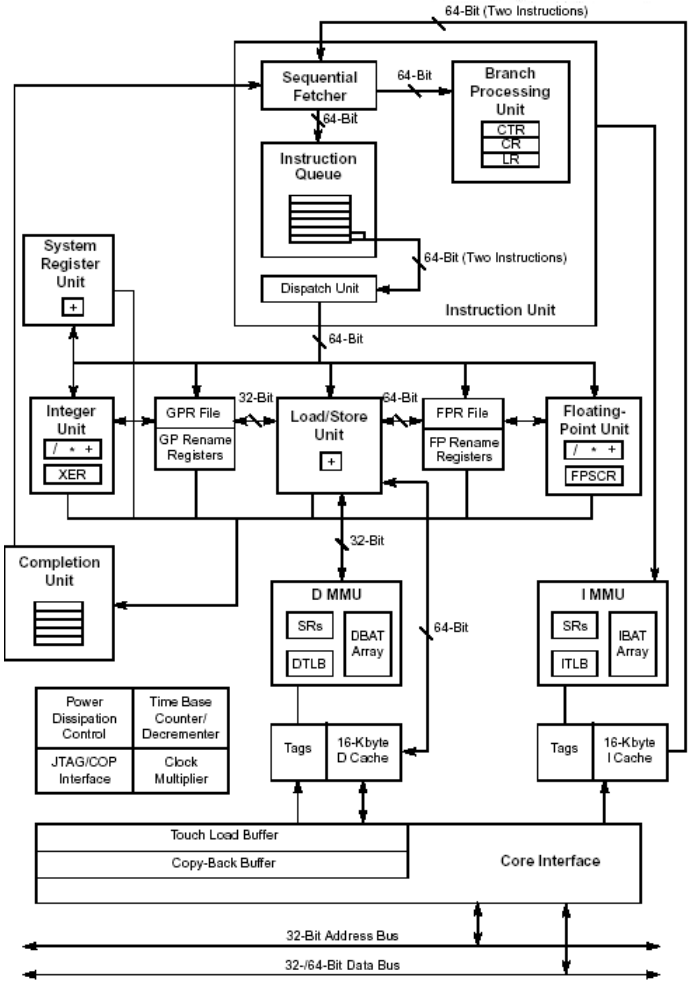

Figura B.2: Arquitetura do processador MPC5200.

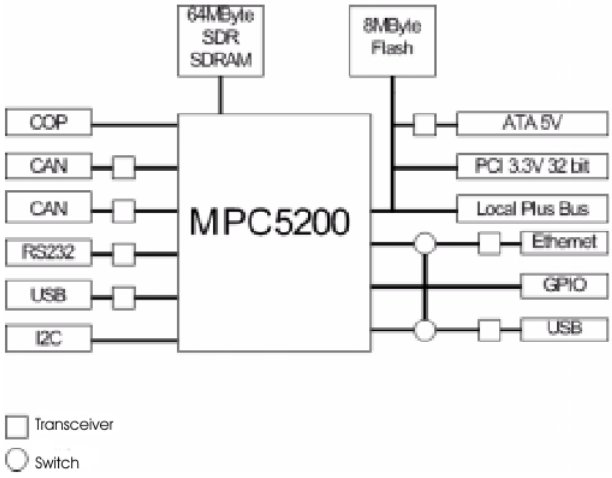

Figura B.3: Periféricos do kit LITE5200.

Referências Bibliográficas

AMIDI, O. (1990). Integrated mobile robot control. Carnegie Mellon University, The Robotics Institute.

AWAD, M., KUUSELA, J., e ZIEGLER, J. (1996). *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT ad Fusion*. Prentice-Hall.

BALL, R. S. (1900). *A Treatise on The Theory of Screws*. Cambridge U. P.

BARRAQUAND, J. e LATOMBE, J. (1989). On nonholonomic mobile robots and optimal maneuvering. *IEEE*, Vol. pp. 340–347.

BECKER, L. B. (2003). *Um Método para Abordar todo o Ciclo de Desenvolvimento de Aplicações Tempo Real*. Tese de Doutorado, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS.

BECKER, L. B. e PEREIRA, C. E. (2002). Simoo-rt - an object-oriented framework for the development of real-time industrial automation systems. *IEEE Transactions on Robotics and Automation*, Vol. 18pp. 421–430.

BERGH, L. G. e MACGREGOR, J. F. (1987). Constrained minimum variance controllers: Internal model structure and robustness properties. *Ind. Eng. Chem. Res.*, Vol. 26pp. 1558–1564.

BERLIN, F. e FRANK, P. (1991). Robust predictive robot control. *IEEE*, Vol. pp. 1493–1496.

BOYDEN, F. e VELINSKY, S. (1994). Dynamic modeling of wheeled mobile robots for high load applications. *IEEE*, Vol. pp. 3071–3078.

BUTTAZZO, G. C. (1997). *Hard Real-time Computing Systems*. Kluwer Academic Publishers.

CAMACHO, E. e BORDONS, C. (1998). *Model Predictive Control*. Springer-Verlag, New York.

CLARKE, D. e MOTHADI, C. (1989). Properties of generalized predictive control. *Automatica*, Vol. 25, No. 6, pp. 859–875.

CLARKE, D. W., MOHTADI, C., e TUFFS, P. S. (1987). Generalized predictive control - part i. the basic algorithm. *Automatica*, Vol. 23, No. 2, pp. 137–148.

COUTLER, R. (1992). Implementation of the pure pursuit path tracking algorithm. Carnegie Mellon University, The Robotics Institute.

CUTLER, C. e RAMAKER, B. (1988). Dynamic Matrix Control - A computer control algorithm. Proc. *AIChE 86th National Meeting*, Houston, TX.

de OLIVEIRA, V. M. (2001). Técnicas de controle de robôs móveis. Dissertação de Mestrado, Univesidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, SC.

DE WIT, C. C., KHENNOUF, H., SAMSON, C., e SØRDALEN, O. J. (1993). *World Scientific Series in Robotics and Intelligent Systems*, chapter Nonlinear Control Design for Mobile Robots, pp. 121–157. World Scientific Publisher.

DEITEL, H. M. e DEITEL, P. J. (2001). *C++: Como Programar*. Bookman.

FRANZON, C. (2004). Kit de desenvolvimento para veículos autônomos. Relatório técnico, Universidade do Vale do Rio dos Sinos, UNISINOS.

GÓMEZ-ORTEGA, J. e CAMACHO, E. F. (1996). Mobile robot navigation in a partially structured static environment using neural predictive control. *Control Engineering Practice, Oxford*, Vol. 4, No. 12, pp. 1669–1679.

GOMES, G. K. (2003). Projeto e implementação de um sistema de controle de velocidade *ACC Stop and Go* para um veículo autônomo. Relatório técnico, Universidade do Vale do Rio dos Sinos, UNISINOS.

GOMES, G. K., KELBER, C. R., SCHIRMBECK, J., BORGES, D. A., RODRIGUES, M. S., e BORCHARDT, I. G. (2000). Projeto de uma bússola eletrônica para aplicações em robótica móvel. *Estudos Tecnológicos - Engenharia*, Vol. 20pp. 15–19.

GOODWIN, G. e SIN, K. (1984). *Adaptive Filtering Prediction and Control*. Prentice Hall.

GU, D. e HU, H. (2002). Neural predictive control for a car-like mobile robot. *Robotics ans Autonomous Systems*, Vol. 39pp. 73–86.

IEEE, C. (2000). Special issue on oo rt distributed computing. *IEEE Computer*, Vol. June.

JUNG, C. R., OSÓRIO, F. S., KELBER, C. R., e HEINEN, F. (2005). Projeto e implementação de veículos autônomos inteligentes. Em da Computação: Um Agente de Inovação e Conhecimento, A. U., editor, *XXV Congresso da SBC - JAI - Jornada de Atualização em Informática.*, pp. 1358–1406.

KELBER, C. R., DREGER, R. S., GOMES, G. K., WEBBER, D., SCHIRMBECK, J., NETTO, R. H., e BORGES, D. A. (2003). Cell phone guided vehicle - an application based on a drive-by-wire automated system. Proc. *IEEE Intelligent Vehicles Symposium*, Munich.

KELBER, C. R., WEBBER, D., GOMES, G. K., LOHMANN, M. A., RODRIGUES, M. S., e LEDUR, D. (2004). Active steering unit with integrated acc for x-by-wire vehicles using a joystick as h.m.i. *IEEE Intelligent Vehicles Symposium*, Vol. s.n.pp. 173–177.

KEYSER, R. D. e CUAWENBERGHE, A. (1985). Extended prediction self adaptive control. Proc. *IFAC Simp. on Ident. and Syst. Parameter Estimation*, pp. 1317–1322, York.

KÜHNE, F. (2005). Controle priditivo de robôs móveis não holonômicos. Dissertação de Mestrado, Univesidade Federal do Rio Grande do Sul, Departamento de Engenharia Elétrica, Porto Alegre, RS.

KÜHNE, F., LAGES, W. F., e da SILVA JR., J. M. G. (2004). Model predictive control of a mobile robot using linearization. *IEEE Mechatronis and Robotics*, Vol. 4pp. s.n. Proceedings...

KIM, B., NECSULESCU, D., e SASIADEK, J. (2001). Model predictive control of an autonomous vehicle. *International Conference on Advanced Intelligent Mechatronics ProceedingsIEEE*, Vol. pp. 1279–1284.

LAGES, W. F. (1998). *Controle e Estimação de Posição e Orientação de Robôs Móveis*. Tese de Doutorado, Instituto Tecnológico da Aeronáutica, Departamento de Engenharia Eletrônica e Computação, São José dos Campos, SP.

LEDUR, D. (2003). Dinâmica de automóveis: Modelo matemático e controle de trajetória. Relatório técnico, Universidade do Vale do Rio dos Sinos, UNISINOS.

MURRAY, R. e SASTRY, S. (1993). Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, Vol. 38, No. 5, pp. 700–717.

NECSULESCU, D., LONMO, V., KIM, B., e DROGUET, E. (1996). Autonomous mobile robot control using kinematics and dynamics based approaches - an experimental analysis. *IEEE*, Vol. pp. 761–765.

NELSON, W. (1989). Continuous steering-function control of robot carts. *IEEE Transactions on Industrial Electronics*, Vol. 36, No. 3, pp. 330–337.

NELSON, W. e COX, I. (1988). Local path control for an autonomous vehicle. *IEEE*, Vol. pp. 1504–1510.

NORMEY-RICO, J. (2003). Controle preditivo de processos com grandes atrasos de transporte. Relatório técnico, Universidade Federal de Santa Catarina, Departamento de Automação e Sistemas. Mini-Curso.

NORMEY-RICO, J. e CAMACHO, E. (2000). Multivariable generalized predictive controller based on the smith predictor. *IEE Proc.-Control Theory Appl.*, Vol. 147, No. 5, pp. 538–546.

NORMEY-RICO, J., GÓMEZ-ORTEGA, J., ALCALÁ-TORREGO, I., e CAMACHO, E. (1998). Low time-consuming implementation of predictive path-tracking control

for a "synchro-drive"mobile robot. *5th International Workshop on Advanced Motion Control*, Vol. pp. 350–356.

NORMEY-RICO, J., GÓMEZ-ORTEGA, J., e CAMACHO, E. (1999). A smith-predictor-based generalized predictive controller for mobile robot path-tracking. *Control Engineering Practice (IFAC)*, Vol. 7pp. 729–740.

OLLERO, A. e AMIDI, O. (1991). O. predictive path tracking of mobile robots: application to the cmu navlab. *IEEE INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS*, Vol. s.n.pp. 1081–1086.

OLLERO, A. e HEREDIA, G. (1995). Stability analisis of mobile robot path tracking. *IEEE*, Vol. s.n.pp. 461–466.

OMG, O. M. G. (2002). Uml profile for schedulability, performance, and time specification. OMG, USA.

PALMOR, Z. (1982). Properties of optimal stochastic control systems with dead-time. *Automatica*, Vol. 18pp. 107–116.

PIAZZI, A., ROMANO, M., e BIANCO, C. G. L. (2004). g^3-splines for the path planning of wheeled mobile robots. Università di Parma, Dipartimento di Ingegneria dell'Informazione.

RAFFO, G. V. (2005). Algoritmos de controle preditivo para seguimento de trajetória de veículos autônomos. Dissertação de Mestrado, Univesidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, SC.

RICHALET, J., el ATA-DOSS, S. A., ARBER, C., KUNTZE, H., JACUBASH, A., e SCHILL, W. (1987). Predictive functional control. Application to fast and accurate robots. Proc. *Proc. 10th IFAC Congress*, Munich.

RICHALET, J., RAULT, A., TESTUD, J., e PAPON, J. (1976). Algorithm control for industrial processes. Proc. *Proc. 4th IFAC Symp. on Identification and System Parameter Estimation*, Tbilisi, URSS.

RUMBAUGH, J. (1991). *Object Oriented Modeling and Design*. Prentice-Hall.

SCHAMMASS, A., VALENTE, C., e CAURIN, G. (1998). Planejamento de trajetória de uma agv utilizando redes neurais. *Proceedings of XII CBA*, Vol. 2pp. 599–603. Brazilian Automatic Control Conference.

SELIC, B., GULLEKSON, G., e WARD, P. T. (1994). *Real-Time Object-Oriented Modeling*. Wiley.

SKRJANC, I. e MATKO, D. (1994). *Advances in Model Based Predictive Control. Chapter: Fuzzy Predictive Controller with Adaptive Gain.*, chapter s.n. Oxford University.

SOETERBOEK, R. (1992). *Predictive Control: A unified approach*. Prentice Hall.

TAN, Y. e KEYSER, R. D. (1994). *Advances in Model Based Predictive Control. Chapter: Neural Network Based Predictive Control.*, chapter s.n. Oxford University.

TOUNSI, M. e CORRE, J. L. (1996). Trajectory generation for mobile robots. *Mathematics and Computers in Simulation*, Vol. 41pp. 367–376.

van ESSEN, H. e NIJMEIJER, H. (2001). Non-linear model predictive control of constrained mobile robots. *EUROPEAN CONTROL CONFERENCE*, Vol. pp. 1157–1162.

WIT, J., III, C. C., e Armstrong, D. (2004). Autonomous ground vehicle path tracking. *Journal of Robotic Systems*, Vol. 21pp. 439–449.

WOLF, W. (2001). *Computers as Components: Principles of Embedded Computing System Design*. McGraw-Hill.

YANG, X., HE, K., GUO, M., e ZHANG, B. (1998). An intelligent predictive control approach to path tracking problem of autonomous mobile robot. *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS*, Vol. s.n.pp. s.n.

YDSTIE, B. (1984). Extended horizon adaptive control. Proc. *9th IFAC World Congress*, Budapest, Hungary.